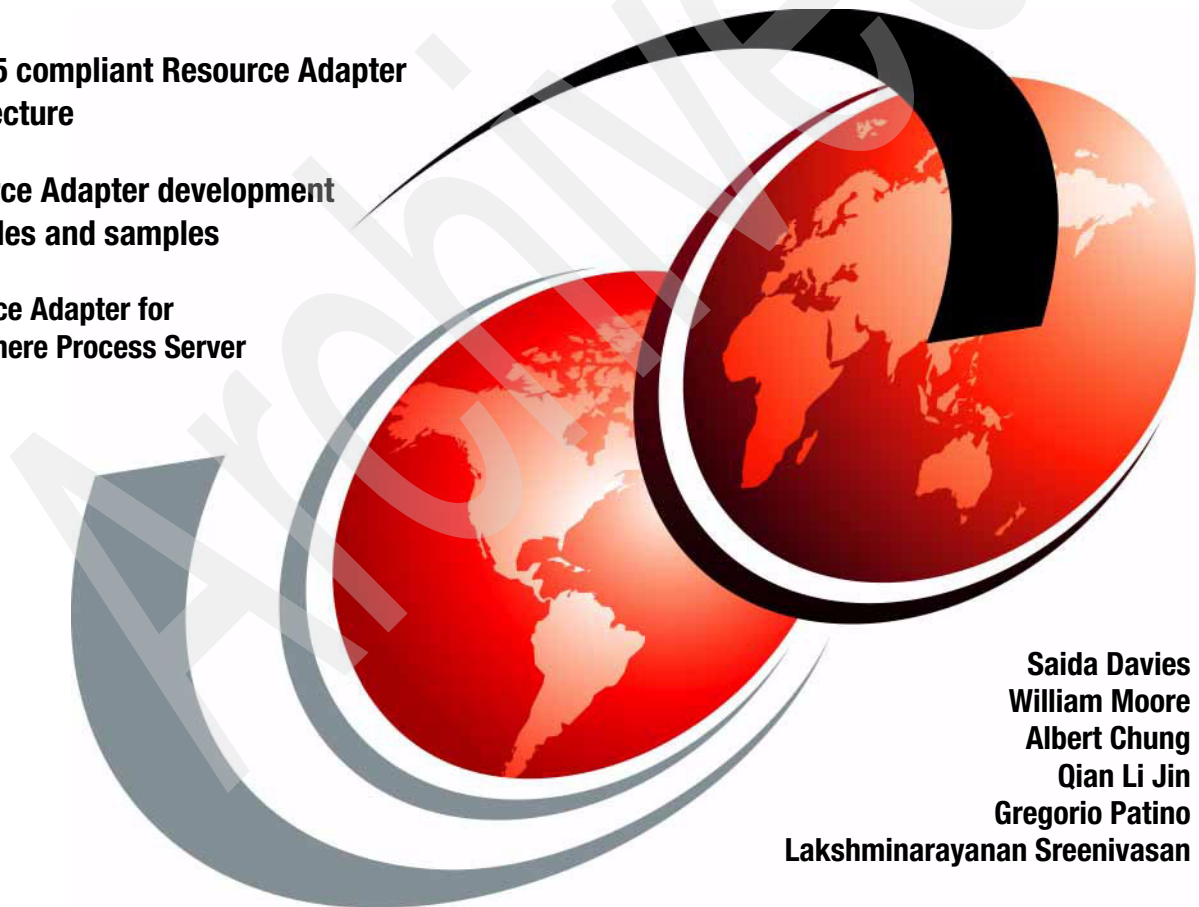


WebSphere Adapter Development

JCA 1.5 compliant Resource Adapter Architecture

Resource Adapter development examples and samples

Resource Adapter for WebSphere Process Server



Saida Davies
William Moore
Albert Chung
Qian Li Jin
Gregorio Patino
Lakshminarayanan Sreenivasan



International Technical Support Organization

WebSphere Adapter Development

June 2006

Archived

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

First Edition (June 2006)

This edition applies to:

Version 6, Release 0, Modification 1 of WebSphere Integration Developer (product number 5724-I66).

Version 6, Release 0, Modification 0, Fix 1 of WebSphere Adapter Toolkit.

© Copyright International Business Machines Corporation 2006. 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

Figures	xi
Tables	xvii
Notices	xix
Trademarks	xx
Preface	xxi
The team that wrote this redbook	xxii
Become a published author	xxiv
Comments welcome	xxiv
Part 1. Adapter development theory	1
Chapter 1. Introduction and adapter overview	3
1.1 Scope and objectives of this book	4
1.2 Introduction to IBM On Demand Business	4
1.2.1 Business attributes	5
1.2.2 Technology attributes	6
1.2.3 Service-oriented architecture	9
1.2.4 Service-oriented architecture and IBM On Demand Business	14
1.2.5 The on demand operating environment and the Enterprise Service Bus	15
1.3 WebSphere Process Server	17
1.3.1 Architectural model	19
1.3.2 Programming model	20
1.3.3 Service Component Architecture	22
1.3.4 Handling data	26
1.3.5 Composition: Business Process Execution Language	26
1.3.6 Other service implementation types	28
1.3.7 For more information	29
1.4 WebSphere Integration Developer and Rational Application Developer ..	29
1.4.1 Integration developer	30
1.4.2 Software (J2EE) developer	31
1.5 Adapters in process integration	31
Chapter 2. Adapter basics	33
2.1 Introduction to adapters	34
2.2 Overview of JCA resource adapters	35

2.2.1 Motivation for JCA adapters	36
2.2.2 J2EE Connector Architecture overview	38
2.2.3 Resource adapter packaging	47
2.3 IBM WebSphere Adapters	47
2.4 WebSphere Business Integration Adapters	50
2.5 IBM WebSphere Adapter versus JCA resource adapter	51
Chapter 3. Service Data Objects	53
3.1 Data: Business Objects and SDO	54
3.1.1 SDO design points	54
3.1.2 Some SDO concepts	57
3.1.3 Business objects and the business object framework	58
3.2 WebSphere Adapter business object model	61
3.2.1 After-images versus deltas	63
3.2.2 Verbs	63
3.2.3 Verbs versus operations	64
3.2.4 Business object naming	64
3.2.5 Blank, Ignore, and Null properties in business objects	65
3.3 Business object structure	65
3.3.1 Single-cardinality relationships	66
3.3.2 Single-cardinality relationships and data without ownership	67
3.3.3 Multiple-cardinality relationships in business objects	67
3.4 Application-specific information (ASI)	69
3.4.1 Application-specific information at the business-object level	69
3.4.2 Application-specific information at the Verb level	69
3.4.3 Application-specific information at the attribute level	70
Chapter 4. Outbound request processing	73
4.1 Outbound request processing	74
4.2 Adapter clients	75
4.3 Service input and output data	76
4.4 Adapter service interface	77
4.5 EIS service import	80
4.6 Adapter configuration properties	82
4.7 Application sign-on	82
4.8 Connection management	83
4.9 Outbound interaction	84
4.10 Standard outbound operations and processing	86
4.10.1 ApplyChanges operation	86
4.10.2 After-Image Create operation	86
4.10.3 After-Image Update operation	88
4.10.4 After-Image UpdateWithDelete operation	89
4.10.5 After-Image Delete operation	90

4.10.6 Retrieve operation	91
4.10.7 RetrieveAll operation	92
4.10.8 Custom operations	94
4.11 Command Pattern	95
4.11.1 Command Pattern processing snapshot objects	96
4.11.2 Command Pattern processing delta objects	98
4.11.3 How to use the Command Pattern	99
4.12 Transaction support	102
Chapter 5. Inbound processing and events	107
5.1 Inbound processing overview	108
5.1.1 Inbound processing flow	109
5.1.2 Inbound components	109
5.2 JCA message inflow contract	111
5.3 Event management	114
5.3.1 Event overview	114
5.3.2 Event management process	116
5.3.3 Event management classes	117
5.3.4 Once and only once delivery	121
5.3.5 Error handling in event management	124
5.4 Service Component Architecture and inbound processing	126
5.4.1 Adapter Clients	126
5.4.2 Service input and output data	126
5.4.3 Adapter service interface	126
5.4.4 EIS service export	128
5.5 Inbound processing development steps	129
5.5.1 Define the inbound properties	130
5.5.2 Implement the connection with the EIS event store	131
5.5.3 Analyze the need for a custom Event class	132
5.5.4 Implement the custom EventStore class	132
5.5.5 Modify the ActivationSpec subclass	134
5.5.6 Modify the ResourceAdapter subclass	134
5.5.7 Implement an event store or not	134
Chapter 6. The theory of Enterprise Service Discovery	137
6.1 EMD definitions	138
6.2 Enterprise Metadata Discovery overview	138
6.3 Architecture	139
6.3.1 J2CA components related to EMD	140
6.3.2 Relationships between components	142
6.3.3 J2EE Roles of Enterprise Metadata Discovery	143
6.4 Process of generating service by using ESD	144
6.5 Process of developing EMD for a resource adapter	145

Chapter 7. Exceptions, logging, and tracing	147
7.1 Adapter exception handling	148
7.1.1 Exception creation during adapter development	148
7.1.2 Exception generation at runtime	149
7.2 Logging and tracing	149
7.2.1 Logging	149
7.2.2 Tracing	151
7.2.3 Message files	152
7.2.4 Business events	154
Part 2. Custom adapter development	157
Chapter 8. Setting up the development environment	159
8.1 Installing WebSphere Integration Developer V6.0	160
8.2 WebSphere Adapter Toolkit overview	174
8.3 Installing WebSphere Adapter Toolkit V6.0.0.1	175
8.4 Create a Hello World adapter	181
8.4.1 Service Component Architecture import for outbound operation	201
8.4.2 Export application EAR file	204
8.4.3 Dependent files	205
8.4.4 Configure J2C authentication alias	207
8.4.5 Install the HelloWorld application	210
8.4.6 Configure server	212
8.4.7 Test HelloWorld adapter	213
8.5 Importing a sample adapter into your development environment	215
Chapter 9. The sample business scenario	221
9.1 Sample scenario overview	222
9.2 Sample integration scenario overview	223
9.3 RedMaintenance application overview	224
9.3.1 Application environment	225
9.3.2 Entity Relationship model	225
9.3.3 Installing and setting up the RedMaintenance application	226
9.3.4 RedMaintenance API	230
Chapter 10. Adapter design and specification	237
10.1 Adapter design technical assessment	238
10.2 RedMaintenance adapter design and specification	241
10.2.1 Architecture overview	241
10.2.2 WebSphere adapter foundation classes	243
10.2.3 RedMaintenance adapter features	246
Chapter 11. Implementing outbound request processing	251
11.1 Start the development project	252

11.2 Identify application properties and operations	253
11.2.1 Connection properties	253
11.2.2 Outbound operations.	253
11.3 Generate outbound stub classes.	255
11.4 Add dependent jar files	261
11.5 Implement connection to EIS	262
11.5.1 Implement RMConnection class	263
11.5.2 Implement RMConnectionFactory class	264
11.5.3 Implement RMManagedConnectionFactory class	264
11.5.4 Implement RMManagedConnection class	266
11.6 Implement outbound operations	268
11.6.1 Implement RMInteraction class.	270
11.6.2 Implement RMInteractionSpec class.	272
11.6.3 Create ObjectNaming utility class	273
11.6.4 Create ObjectConverter utility class	276
11.6.5 Implement RMCommandFactoryImpl class	279
11.6.6 Implement RMBaseCommand class.	281
11.6.7 Implement RMCreateCommand class	283
11.6.8 Implement RMUpdateCommand class	285
11.6.9 Implement RMRetrieveCommand class	287
11.6.10 Implement RMDeleteCommand class.	290
11.6.11 Implement RMResourceAdapter.	293
11.6.12 Implement outbound log messages	295
11.6.13 Update MANIFEST.MF file	297
11.7 Create temporary SCA artifacts	297
11.7.1 Create business objects with application-specific information	298
11.7.2 Creating RedMaintenance ASI schema	311
11.7.3 Create discovery-service.xml file	313
11.7.4 Create Apartment business object graphs	314
11.7.5 Create RedMaintenance outbound interface	315
11.7.6 Create SCA EIS service import file	317
11.7.7 Exporting the adapter	318
11.8 Test outbound operations	320
11.9 Configure logging and tracing levels for debugging	336
Chapter 12. Implementing inbound processing and event handling	343
12.1 Generate the inbound stub classes.	344
12.1.1 Start a new adapter project.	344
12.1.2 Using an existing adapter project	345
12.2 Identify inbound events and properties	347
12.2.1 Inbound events	347
12.2.2 Inbound properties	348
12.3 Add dependent jar files	355

12.4	Connection to EIS for inbound processing	355
12.5	Implement the RMEvent subclass	356
12.6	Implementation of RMEventStore class	358
12.6.1	Event filtering	359
12.6.2	Polling events	359
12.6.3	Deleting events	360
12.6.4	Updating events	361
12.6.5	Retrieving events	362
12.6.6	Retrieving business objects	363
12.6.7	Transactional Considerations	364
12.6.8	Full code of RMEventStore	365
12.7	Revision of the RMActivationSpec class	376
12.8	Change of RMResourceAdapter class	377
12.9	Creation of the SCA artifacts	377
12.9.1	Provide the basic business objects and related files	378
12.9.2	Create the RedMaintenance inbound interface	378
12.9.3	Create the SCA EIS export file	379
12.9.4	Create the SCATestComponent	381
12.9.5	Exporting the adapter	385
12.10	Test inbound operations	385
12.10.1	Sample stand-alone RedMaintenance test client	386
12.10.2	Setup integration test client	388
12.10.3	Run stand-alone RedMaintenance test client	389
12.10.4	Observe the effects of these inbound events	390
12.11	Configure logging and tracing levels for debugging	392
Chapter 13.	Implementing Enterprise Metadata Discovery	393
13.1	Analyze EIS metadata before EMD implementation	394
13.2	Use WebSphere Adapter Toolkit to generate EMD stub classes	394
13.2.1	Start a new adapter project	395
13.2.2	Using an existing adapter project	399
13.2.3	Create the deployment descriptor for EMD	400
13.2.4	Application-specific information schema	401
13.2.5	Understand utility APIs provided by EMD tooling	403
13.2.6	Implement EMD stub classes	404
13.2.7	Testing the EMD implementation by running it in WID	420
Appendix A.	Additional material	435
	Locating the Web material	435
	Using the Web material	435
	How to use the Web material	436

Abbreviations and acronyms 437

Related publications 439

IBM Redbooks 439

Online resources 439

How to get IBM Redbooks 439

Help from IBM 440

Index 443

Archived

Figures

1-1	e-business on demand overview diagram	5
1-2	Four technology attributes of IBM On Demand Business	7
1-3	SOA as a new architectural approach	10
1-4	Example of service granularity and choreography	12
1-5	On demand operating environment based on SOA	15
1-6	Business-driven development process and related tools	18
1-7	Architectural model of WebSphere Process Server	19
1-8	Programming model: today	21
1-9	Programming model: simplification	22
1-10	Service component: overview	23
1-11	Service component and references	25
1-12	Service module: overview	26
1-13	BPEL Process example	28
1-14	Business Driven Development: Roles and Tools	30
1-15	EIS integration using adapters	31
1-16	The role of the adapter in WebSphere Process Server	32
2-1	Electric plugs of other countries	34
2-2	Adapter pattern	35
2-3	Complexities of point to point integration	36
2-4	Application servers using resource adapter to interact with EIS 1	37
2-5	Application server with multiple EISs using a resource adapter	37
2-6	Resource adapter deployed to an application server	38
2-7	JCA system contract: application server and resource adapter	39
2-8	Adapter connection management architecture	40
2-9	Transaction management	42
2-10	Security management	43
2-11	IBM WebSphere Adapter	48
2-12	WebSphere Adapter architecture	49
2-13	IBM WebSphere Business Integration Adapter	50
3-1	Data in SOA: Service Data Objects	54
3-2	Data graph consisting of data objects and change summary	58
3-3	Exchanging data in an SCA runtime	59
3-4	Business graph and business objects	61
3-5	WebSphere Adapter Business Object model	62
3-6	Typical single cardinality relationship	67
3-7	Typical multiple cardinality relationship	68
3-8	Multiple cardinality relationship with N=1	68
4-1	Connecting to EIS through EIS import and adapter	75

4-2	Collection of business objects	93
4-3	Command Manager command structure for after-image CREATE	97
4-4	Command Manager command structure for after-image Update	98
4-5	Command Manager command structure for delta CREATE	99
4-6	Local Transaction	103
4-7	XA Transaction	103
5-1	Inbound processing flow	109
5-2	Inbound components	110
5-3	Message Inflow actors	112
5-4	Message inflow sequence diagram	113
5-5	Event management	116
5-6	Event management class diagram	118
5-7	Event management: step 1	122
5-8	Event management: step 2	123
5-9	Event management: step 3	123
5-10	Event management: step 4	124
6-1	Enterprise Service Discovery architecture	140
6-2	Enterprise Service Discovery roles	143
6-3	Using Enterprise Service Discovery to create a service	144
8-1	Setup launchpad	160
8-2	Welcome dialog	161
8-3	License dialog	162
8-4	Installation location directory dialog	163
8-5	Feature selection dialog 1	164
8-6	Feature selection dialog 2	165
8-7	Pre-install summary dialog	166
8-8	Installation dialog	167
8-9	Post install summary dialog	168
8-10	Readme dialog	169
8-11	Finish dialog	170
8-12	Launch Rational Product Updater from installer	171
8-13	Rational Product Updater	172
8-14	Available updates	173
8-15	License agreement dialog for updates	174
8-16	Welcome	176
8-17	License agreement	177
8-18	Install location directory	178
8-19	Preinstallation summary	179
8-20	WebSphere Adapter Toolkit installation	180
8-21	Post-installation summary	181
8-22	Hello World workspace	182
8-23	New project dialog	183
8-24	New J2C Resource Adapter Project dialog	184

8-25	Adapter properties	185
8-26	Select components to be generated by the toolkit	186
8-27	Switch to J2EE perspective	186
8-28	Generated files	187
8-29	Business Integration view's context menu	190
8-30	New Module dialog	191
8-31	Dependency editor	192
8-32	Select dependent J2EE project	193
8-33	Open new business object dialog	194
8-34	Add name attribute to Person business object	195
8-35	Generate business graph	196
8-36	Create new interface	197
8-37	Enter new interface name	198
8-38	Create new outbound operation	199
8-39	Add input parameter to outbound operation	200
8-40	Add output parameter to outbound operation	201
8-41	Create a new file	202
8-42	HelloWorld adapter import	204
8-43	HelloWorldModuleApp deployment descriptor	205
8-44	Open WebSphere Process Server administrative console	206
8-45	Configure J2C Authentication data	208
8-46	Create new authentication alias	209
8-47	Install Hello World application to WebSphere Process Server	210
8-48	Install Hello World application (continued)	211
8-49	Start Hello Wold application	212
8-50	Open Test Component tool to test Hello World adapter	213
8-51	Enter parameters for the request business object	214
8-52	Response from HelloWorld adapter	215
8-53	Import TwineBall Sample	216
8-54	Properties for TwineBall Sample	217
8-55	Preference dialog box	218
9-1	Current call center environment	222
9-2	New integration with WebSphere Process Server and adapters	224
9-3	RedMaintenance E-R model	225
9-4	Import script in DB2 Command Center	227
9-5	RedMaintenance data sample	228
9-6	RedMaintenance Shortcut	229
9-7	GUI log	230
9-8	Relationship between classes	233
9-9	RedMaintenance application business objects	234
9-10	Methods of the Event class	235
10-1	RedMaintenance resource adapter architecture	243
10-2	Generic implementation for inbound and outbound	244

11-1	Examples of CRUD operations on Tenant and Worker entities	254
11-2	Adapter Deployment Descriptor editor	257
11-3	Add outbound properties	258
11-4	Add server URL property	259
11-5	Add security permission for the adapter	260
11-6	Main connection classes of the resource adapter	263
11-7	Apartment interface	316
11-8	Start WebSphere Process Server	321
11-9	RedMaintenance outbound interface J2C authentication alias	322
11-10	J2C authentication alias on WebSphere Process Server	323
11-11	Install RedMaintenance module application	324
11-12	createApartment outbound operation	326
11-13	Result of createApartment operation	327
11-14	test retrieveApartment operation	328
11-15	Result of retrieveApartment operation	329
11-16	Test updateApartment operation	331
11-17	Result of the updateApartment test	332
11-18	Test deleteApartment operation	334
11-19	Result of deleteApartment operation	335
11-20	Troubleshooting	336
11-21	Log and Trace	337
11-22	Server selection	338
11-23	Change Log Detail Levels	339
11-24	Set log land trace levels	340
12-1	Adding inbound processing to an existent J2C adapter project	346
12-2	Selecting the Inbound adapter classes	347
12-3	Inserting the custom inbound properties	349
12-4	RedMaintenance Inbound interface	379
12-5	Creating a Component with no implementation type	382
12-6	Wiring the Inbound interface with the SCATestComponent	383
12-7	Events indicating RedMaintenance adapter is polling.	386
12-8	Import RedMaintenance client	387
12-9	Integration test client	388
12-10	Run dialog	389
12-11	Inbound events on RedMaintenance application	390
12-12	Inbound business objects in WebSphere Process Server	391
13-1	Create a new adapter project	396
13-2	New project dialog	397
13-3	Generation options dialog	398
13-4	EMD stub classes generated by WebSphere Adapter Toolkit	399
13-5	Adding inbound processing to an existent J2C adapter project	400
13-6	Create new Enterprise Service Discovery	420
13-7	Select Enterprise Service Resource Adapter	421

13-8	Configure settings dialog	422
13-9	Discovery dialog box	423
13-10	Configure Objects	424
13-11	Generate artifacts dialog	425
13-12	ESD generated service artifacts and business object definition	426

Archived

Tables

3-1	Comparing data-oriented Java APIs	55
5-1	Basic attributes of an event record in an EIS.....	115
5-2	Possible event status values	115
5-3	Events error handling.....	125
5-4	ActivationSpec reserved and standard properties	130
5-5	Inbound custom properties	131
7-1	Logging levels	150
7-2	Trace Message Levels.....	151
9-1	Allowed values for status attribute in the RedMaintenance ASBOs ..	236
12-1	Inbound custom properties	348

Notices

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:


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

Trademarks

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

CICS®
Cloudscape™
DB2 Universal Database™
DB2®
developerWorks®
e-business on demand™

eServer™
@server®
@server®
IBM®
IMS™
Parallel Sysplex®

Rational®
Redbooks (logo) ™
Redbooks™
WebSphere®
z/OS®

The following terms are trademarks of other companies:

Enterprise JavaBeans, EJB, Java, Java Naming and Directory Interface, Javadoc, JavaBeans, JDBC, 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.

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

Preface

This IBM® Redbook shows you how to develop a JCA resource adapter based on IBM WebSphere® Adapter Architecture. The custom adapter described in this book implements the Java™ 2 Enterprise Edition (J2EE™) Connector architecture (JCA), version 1.5, also known as resource adapters or JCA adapters. This adapter also supports managed, bidirectional connectivity between Enterprise Information Systems (EISs) and J2EE components supported by WebSphere Process Server.

J2EE Connector Architecture (JCA) is Java-based solution to connect application servers to Enterprise Information Systems (EIS). It is used to provide Enterprise Application Integration (EAI) solutions. JCA standardizes the way J2EE application components, J2EE compliant application servers and EIS resources interact with each other.

The first part of this book discusses the components that make up a WebSphere Adapter. The second part illustrates how to implement this adapter that fits in a sample integration scenario.

WebSphere Adapter Architecture supports service-oriented architecture (SOA). SOA provides you with the ability to develop and modify integration applications dynamically. It also lets you integrate existing applications with newer applications, so that they work together transparently.

WebSphere Process Server is a comprehensive SOA integration platform and is based on WebSphere Application Server V6. You can use WebSphere Process Server to develop and execute standards-based, component-based business integration applications in a SOA.

WebSphere Process Server supports two capabilities required for an efficient service-oriented architecture:

- ▶ A universal invocation model, implemented as a Service Component Architecture (SCA)
- ▶ A universal data representation implemented, as business objects (BOs).

This book describes how to create a custom resource adapter that leverages the services provided by WebSphere Adapter Foundation Classes. We also show you how a this adapter can be exposed as an SCA service and used with other SCA components running on WebSphere Process server V6.0.1.

A WebSphere Adapter does the following:

- ▶ Integrates with WebSphere Process Server.
- ▶ Connects an application running on WebSphere Process Server with an EIS.
- ▶ Enables data exchange between the application and the EIS.

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, Raleigh Center.



Saida Davies is a Project Leader for the International Technical Support Organization (ITSO) and has seventeen years of experience in IT. She has published several IBM Redbooks™ on various WebSphere Business Integration topics. Saida has experience in the architecture and design of WebSphere MQ solutions, extensive knowledge of the IBM z/OS® operating system and a detailed working knowledge of both IBM and Independent Software Vendors' operating system software. In a customer-facing role as a senior IT specialist with IBM Global Services, her role included the development of services for z/OS and WebSphere MQ within the z/OS and Windows® platform. This covered the architecture, scope, design, project management and implementation of the software on stand-alone systems or on systems in a Parallel Sysplex® environment. she has received Bravo Awards for her project contributions. She has a degree in Computer Studies and her background includes z/OS systems programming. Saida supports Women in Technology activities, and contributes and participates in the their meetings.



William Moore is a technical staff member at the IBM Extreme Blue lab in Raleigh where he provides project coordination and leadership for Extreme Blue projects.

From 2000 to 2006 Bill was a WebSphere specialist at the ITSO, Raleigh Center producing IBM Redbooks and other associated documentation for Application Integration as well as Middleware and Rational products. He wrote extensively and taught classes on WebSphere and related topics. Before joining the ITSO, Bill was a Senior AIM Consultant at the IBM Transarc lab in Sydney, Australia. He has 21 years of application development experience on a wide range of computing platforms using many different coding languages. His current areas of expertise include J2EE, WebSphere Application Server, application development tools, object-oriented programming and design, and e-business application development.



Albert Chung is a Software Engineer based at IBM Research Triangle Park Lab, Raleigh, North Carolina. He joined IBM in 2001 and currently works as the WebSphere Adapters installation development team lead. He holds a degree in Electrical Engineering from University of Technology, Jamaica and a degree in Computer Science from Florida International University, USA. His areas of expertise include software installation and J2EE.



Qian Li Jin is a currently working as a software engineer at IBM China, Software Development Lab. His job mainly focuses on the development of Application Integration Adapters. He is interested in the information integration and data mining.



Gregorio Patino is a Consulting IT Architect at PRAGMA, in Medellin, Colombia. He currently advises the top Colombian companies in IT strategies and software project development. His areas of expertise include J2EE development, distributed systems, EAI and SOA and Identity Management. He has also taught postgraduate courses at the EAFIT University in these subjects for the past four years.



Lakshminarayanan Sreenivasan (S.L.) is a Software Engineer at IBM Bangalore, India. He has four years of experience in Information Technology, working in various SDLC stages. He has been with IBM for two years and his areas of expertise include Testing IBM Websphere Adapters. He has written extensively on Testing and Exception handling.

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

- ▶ Olga Treyger
WebSphere Adapters, Burlingame Lab, California
- ▶ Corville Allen
WebSphere Adapters, Research Triangle Park Lab, North Carolina
- ▶ Travis Nelson
WebSphere Adapters, Burlingame Lab, California

- ▶ Suraksha Vidyarthi
WebSphere Adapters, Burlingame Lab, California

Become a published author

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our 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. HZ8 Building 662
P.O. Box 12195
Research Triangle Park, NC 27709-2195



Part 1

Adapter development theory

ARCHIVED

Introduction and adapter overview

This chapter discusses the scope of this redbook and lists the order in which the topics are presented in the following chapters, illustrating the strategic vision for IBM On Demand Business. Here, you can find an overview of the key technology and architectural concepts.

The following topics are covered:

- ▶ Scope and objectives of the book
- ▶ Introduction to IBM On Demand Business
- ▶ WebSphere Process Server
- ▶ WebSphere Integration Developer and Rational Application Developer

The second topic discusses the general motivation of the ever-present relationship between business goals and IT goals in an evolutionary business scenario. It also provides a general overview of service-oriented architecture (SOA) as the glue between the business and IT worlds.

The next topic provides a high-level description of WebSphere Process Server, the primary product in the implementation of service-oriented architectures (SOAs) and the product responsible for providing the execution environment for SOA artifacts (custom WebSphere Adapters are a special artifact in integration).

Finally it introduces the tools involved in the development process of adapters.

1.1 Scope and objectives of this book

The objective of this book is to provide a theoretical and practical basis to the developers who design and build custom WebSphere Adapters.

As the IT environments are changing rapidly, the interaction between earlier applications and new technologies like components, services, and the emerging business process automation initiatives are becoming more and more challenging whereby, companies do not have the time and money to rewrite their old applications, so it is necessary to integrate these *old* applications with the *new* demanding environments.

Adapters play a key role in the integration of applications and data using open standards. One aspect of today's business integration requirements is the need to leverage existing Enterprise Information System (EIS) assets in a heterogeneous, integrated enterprise. Whether the business need is to share business data across disparate Enterprise Information System (EIS) assets, automate business processes across EISs, employees, and business partners, connect new e-business applications to existing EISs, or provide users with access to EIS business data, there is a common requirement to provide interfaces to EISs.

This book shows you how to develop an adapter that conforms to the Java Platform, Enterprise Edition (J2EE) Connector Architecture (JCA) standard and is deployable to WebSphere Process Server. This book is divided in two parts. Part 1 explains the theory of adapter development, enabling you to understand the environment and concepts for developing a custom Adapter. This theory begins describing the elements related to the SOA environment and the new challenges in the software development process. The following chapters in this part describe the basic concepts of WebSphere Adapters technology and the explanation of the interaction between the adapter, the earlier application, and the WebSphere Process Server.

Part 2 of the book focuses on the implementation of a custom adaptor using the WebSphere Adapter Toolkit, describing the steps to create a custom Adapter that can be deployed to WebSphere Process Server. During the development process, hands-on experience is also described for using WebSphere Integration Developer.

1.2 Introduction to IBM On Demand Business

The vision of IBM On Demand Business is to enable customers to succeed in an environment with an unprecedented rate of change. Businesses want to focus on core competencies, reduce spending, and reuse existing information in new

ways without a major overhaul of their existing infrastructure. There exists a constant pressure to juggle often conflicting demands to provide flexibility, cost savings, and efficiency.

Figure 1-1 identifies the key components of e-business on demand™.

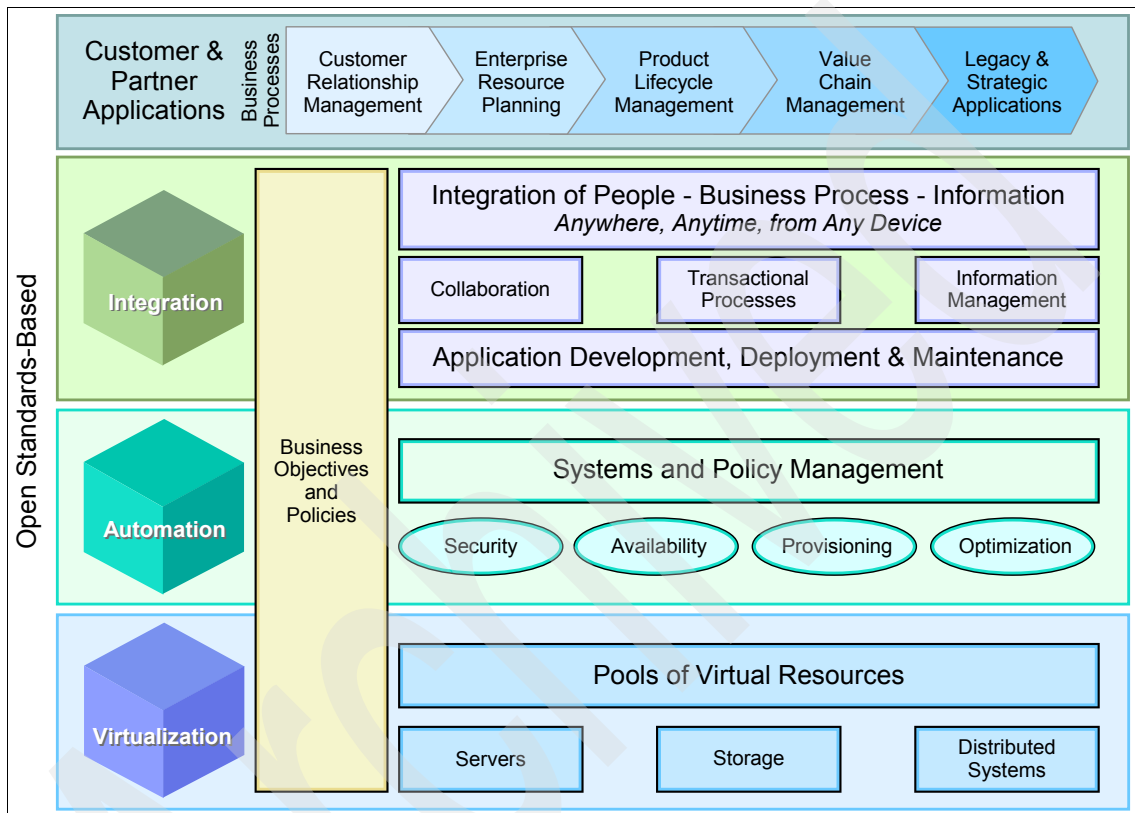


Figure 1-1 e-business on demand overview diagram

The following sections outline the key business and technical attributes that provide the basis for the on-demand message.

1.2.1 Business attributes

From a business perspective, IBM On Demand Business is about providing a way for companies to realign their business and technology environment to match their requirements for reusable business functionality.

The business drivers for IBM On Demand Business can be summarized as:

- ▶ Focused
Enabling the enterprise to focus on their core competencies, what makes them successful and what makes them unique. Strategic alliances are formed to provide needs external to these core competencies.
- ▶ Responsive
The ability to respond with agility to customer demands, market opportunities, or external threats, these decisions are guided through insight-driven decision management features.
- ▶ Variable
To achieve operational and business process flexibility. To adapt variable cost structures (fixed to variable) to provide a high level of operational efficiency.
- ▶ Resilient
Capability and robustness to respond to changes in both business and technical environments, resulting in managed changes and threats with predictable outcomes.

Companies can achieve these business imperatives by exploiting current technological developments while drawing on experiences that have been learned from past architectural constructs.

1.2.2 Technology attributes

The business drivers of e-business on demand must be supported by a well-defined technical infrastructure. These key technological attributes deliver the flexibility, responsiveness, and efficiency that on-demand organizations require:

- ▶ Integration
- ▶ Virtualization
- ▶ Automation
- ▶ Open standards

Figure 1-2 provides a high-level overview of the range of each e-business on demand attribute.

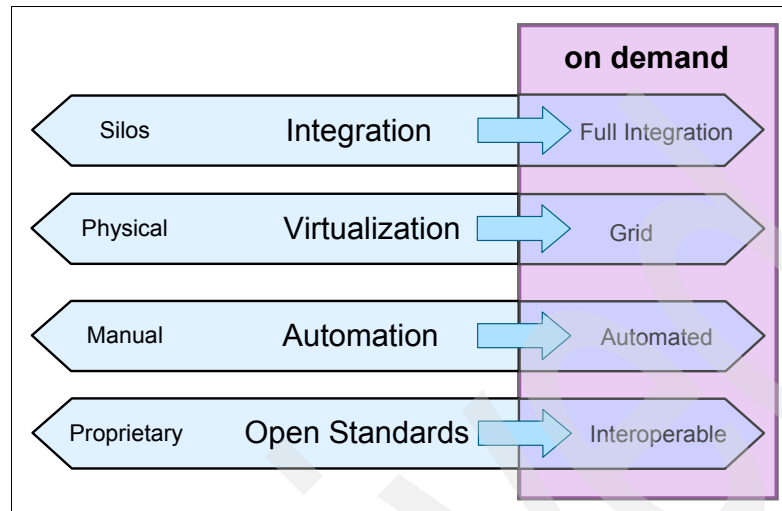


Figure 1-2 Four technology attributes of IBM On Demand Business

Integration

Integration is a technology attribute that can occur between:

► People

The human-to-human and human-to-process interaction must be achieved not only for end users, but customers, employees, as well as trading partners, placing the focus on the promotion of a collaborative environment.

► Processes

Modern business management techniques are focused in modeling on-demand organizations as a set of interrelational, interactive, and automated processes, where information and activities can be arranged and rearranged in a way that permits the steady increase of revenue and the steady reduction of the overall costs.

► Applications

Many organizations have invested enormous resources and capital in custom-designed and off-the-shelf applications. The application integration goal is to leverage, rather than replace, these assets by providing ways of connecting, routing, and transforming the data that is stored or shared among them. Applications can sit on disparate systems in an enterprise or across many enterprises.

- **Systems**

Systems manage, process, and deliver data to the people and applications in the solution environment. An on-demand operating environment requires the system to be transparent to the elements that interact with it.

- **Data**

Data is the primary business element of a system. The data is the source of the information and can be shared more easily through the adoption of standards specifications.

Virtualization

In on-demand environments, the concept of virtualization means that there must be a flexibility in the awareness of the physical location of computational resources. Moreover, users, services, and processes do not need to know how many servers, storage devices, or distributed systems support their requests.

The vision is to create virtual, dynamic organizations through secure, coordinated resource sharing among individuals, institutions, and resources. *Grid computing* is an approach to distributed computing that spans locations, organizations, machine architectures, and software boundaries.

Automation

Automatic computing can be summarized using the four key components:

- **Self-healing**

Self-healing is a system's ability to keep functioning. In order to achieve this, the system must detect, prevent, and recover from disruptions with minimal or no human intervention. This requirement is directly proportional to increased business dependence on technical infrastructures. The need for self-healing is directly proportional to the organization's availability requirement.

- **Self-configuring**

Self-configuring is the ability to adapt dynamically to changing environments, add and remove components to and from the systems, and change the environment to adapt to variable workloads.

- **Self-optimization**

Self-optimization can be described as the configuration that maximizes operational efficiency, including resource tuning and workload management. This alleviates the constant drain on resources to perform routine tasks. The goal is to tune systems to respond to the workload changes. Systems have to monitor and self-tune continuously, adapting and learning from the environment around them.

► Self-protecting

Security is one of the inhibitors of the adoption of SOAs as organizations prepare themselves to share data externally. Self-protection requires the system to provide safe alternatives to secure information and data. Self-protecting automation works by anticipating, detecting, identifying, and protecting systems from external or internal threats.

Open standards

Open standards are the key element of flexibility and interoperability across heterogeneous systems. The global adoption of a standard specification enables disparate systems to interact with each other. While the underlying platforms can be completely different and independent, open standards enable processes to be built despite (or because of) these differences.

In this way, on-demand organizations can adopt changes rapidly into their IT infrastructure and applications, and can interact easily with trading partners, suppliers, and customers.

1.2.3 Service-oriented architecture

Service-oriented architecture (SOA) defines integration architectures designed to provide integrated IT solutions based on the principles of loose coupling and encapsulation. SOA as an architectural approach is new but it is based on many concepts proved by Object Oriented Analysis and Design, Component Based Development, and Enterprise Application Integration technology.

Service

The key concept in this approach is the service. A *service* is a reusable artifact that represents a related group of functions defined by an explicit interface. The interface is implementation-independent, allowing the implementation of the service to change without affecting the way the service is called.

Services are loosely bound and invoked through standard communication protocols, assuring location transparency and interoperability between them. The use of standard protocols enables services in heterogeneous environments to communicate and eases the integration of disparate applications.

These features are summarized in Figure 1-3.

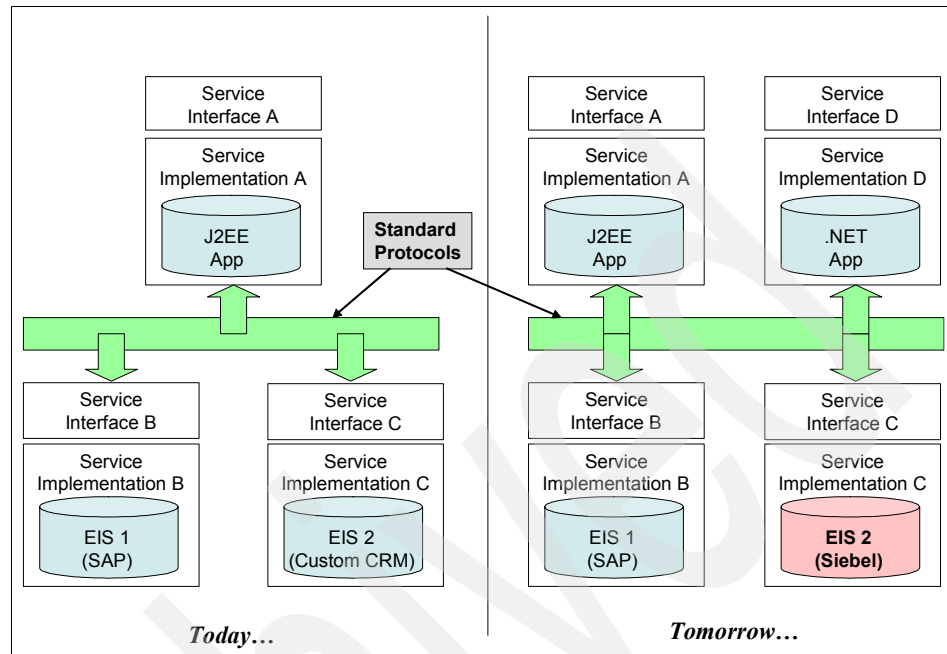


Figure 1-3 SOA as a new architectural approach

Service granularity

The first challenge when adopting an SOA is to define the abstraction level and granularity of the services. Many descriptions of SOA refer to the use of large-grained services, but there are many other examples of fine-grained services that can be used. Common levels of granularity used are:

- Technical functions

Examples of this category are functions such as authentication, authorization, logging, PDF file generation and so on. They represent functionality that can be reused by almost every system in the architecture.

- Business functions

Examples of this category are functions such as `getBalance`, `validateCreditCard` or `getMortgageRate`. These functions often represent business or validation rules imposed by some regulatory agency or market conditions.

- ▶ Business transactions

Examples of this category are functions such as `openAccount`, `updateStockInfo`, or `trackOrder`. The functionality is more likely to be associated with an implementation in a traditional system such as SAP or Peoplesoft.

- ▶ Business processes

Examples of this category are functions such as `applyForMortgage`, `solveCustomerRequest`, `createBankAccount`, or `generateAccountStatements`. These represent a group of activities tied together and involve the execution of tasks in different EIS or human intervention to reach their goal.

Service choreography

Some degree of choreography or aggregation is required between the different levels of granularity, but this choreography must be designed to fit the environment of the company. Figure 1-4 on page 12 shows an example of service granularities and choreographies between them.

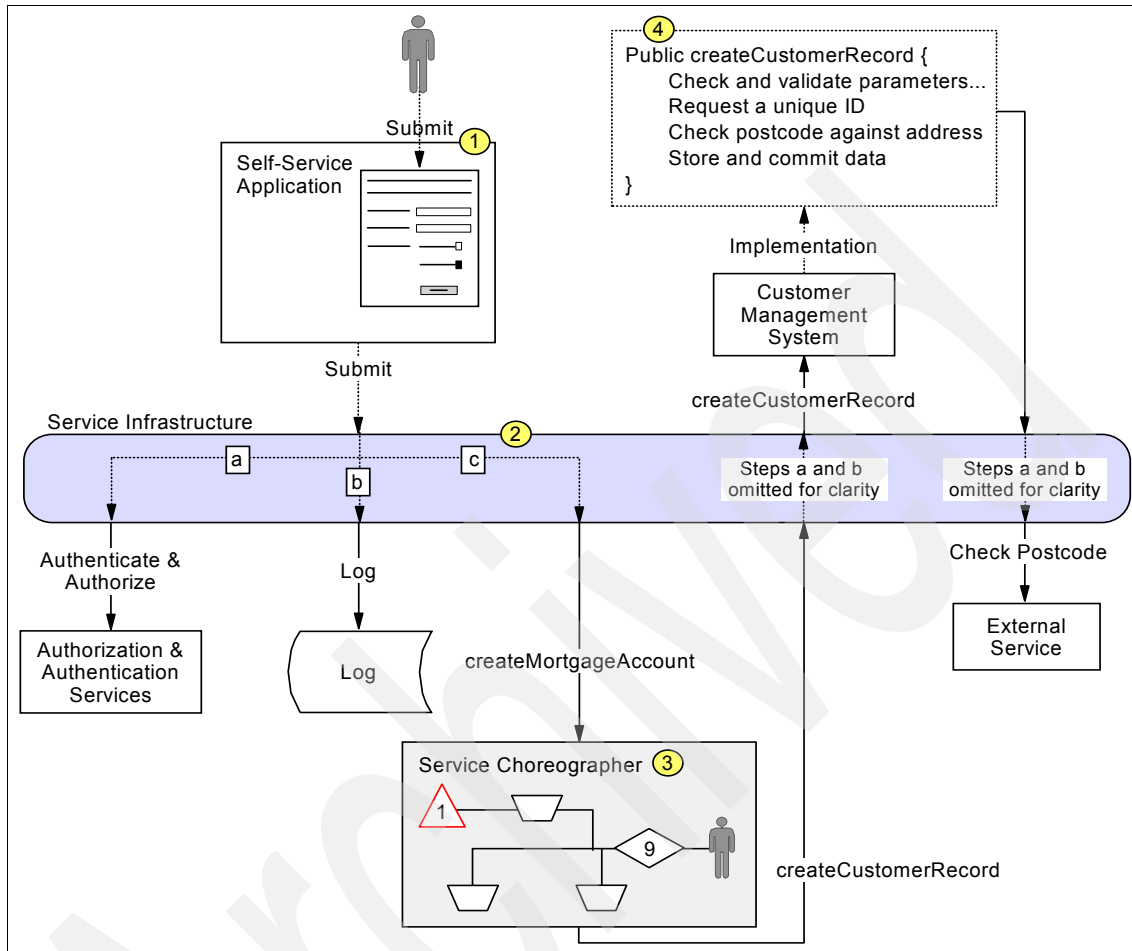


Figure 1-4 Example of service granularity and choreography

In Figure 1-4 you can see the following steps of interactions among services of various granularities:

1. A user submits a request to a self-service application to create a mortgage account. The self-service application submits the business process service request createMortgageAccount through the service infrastructure to a service choreographer component, the purpose of which is to choreograph business transaction services into business process services.
2. On receiving the request for the createMortgageAccount business process service, the service infrastructure first invokes authentication and authorization technical function services to ensure that the request is valid,

then logs a technical function service before finally invoking the createMortgageAccount business process service in the service choreographer.

3. The service choreographer executes the createMortgageAccount business process service. If the request is valid, then when the other process elements are complete, the choreographer invokes the createCustomerRecord business transaction service through the service infrastructure to store the details of the new customer. (Before doing this, it might have invoked storeMortgageDetails previously.)
4. In the implementation of the Customer Management System createCustomerRecord business transaction service, it is necessary to validate the information for the new customer. Part of this validation is checking whether the post code and address match. In order to do this, a CheckPostCode business function service is invoked through the service infrastructure.

To summarize, three aggregations or choreographies are performed by distinct components for distinct granularity levels:

- ▶ Service choreographer
Choreographs business transaction services into higher level business process services.
- ▶ Service infrastructure (can be an enterprise service bus)
Choreographs technical function services to control the invocation of business process services, business transaction services, and business function services.
- ▶ Individual application components
Responsible for invoking business function services where they are required in order to implement business transaction services.

This is just one hypothetical example. Real organizations must formulate their own definitions.

Implications of service-oriented architecture

The adoption of SOA can be a real challenge for companies and must have the commitment of the organization as well as the IT staff. There are many issues that have to be resolved in order to take advantage of a full, service-oriented architecture, for example:

- ▶ Skilled architects, designers, and business analysts are needed to guarantee the correct definition of processes, transactions, and functions.
- ▶ The ownership of data and processes must be established.

- ▶ The boundaries of participating systems have to be clearly defined, and the redundant functionality should be eliminated.
- ▶ The integration of traditional and existing systems instead of the development of new ones and the incremental approach for the implementation of SOA are fundamental in order to show results to the organization.
- ▶ No specific technologies are ruled in or ruled out.
- ▶ The technology is still evolving. It makes the adoption of standards the key for the long term success.

1.2.4 Service-oriented architecture and IBM On Demand Business

Consider how SOA can help you meet the four key technology attributes of an IBM On Demand Business:

▶ Open standards

The use of Web Services is well-suited to implementing an SOA. The open standards associated with Web Services provide a set of flexible and interoperable standards for distributed systems. These standards include:

- Standards for messaging protocols, such as Simple Object Access Protocol (SOAP)
- Standards for transport protocols, including HyperText Transfer Protocol (HTTP), HyperText Transfer Protocol Secure (HTTPS), or Java Message Service (JMS)
- Standards describing language for interfaces of services such as Web Services Description Language (WSDL)

Standard bodies, including Web Services Interoperability (WS-I), World Wide Web Consortium (W3C) and OASIS, with the active participation of technology industry leaders (IBM, BEA, Oracle, Microsoft® and so forth), accelerate and guide the open standards creation and adoption.

The use of open standards permits long-term investment in this architecture, with the knowledge that these standards are supported and evolve in a coordinated way.

▶ Integration

Interfaces are provided to encapsulate different EIS functionalities and to exploit the main features of a heterogeneous IT environment. SOA permits the integration of these interfaces through dynamic discovery and binding to the services, and with the promotion of services choreography.

Moreover, Web Services and SOA are more suitable for integration projects than other proprietary technologies, like Remote Procedure Call (RPC),

COM/COM+, Common Object Request Broker Architecture (CORBA) or even Remote Method Invocation/Internet Inter-ORB Protocol (RMI/IIOP).

- ▶ Virtualization

SOA supports virtualization through the encapsulation of the services implementation which provides location transparency. This makes it easy to deploy and manage services and hides the complexities of the IT infrastructure to the service consumer.

- ▶ Automation

The adoption of services and SOA is the first step in autonomic computing. Grid computing services are being developed to provide an evolutionary approach to increased automation.

1.2.5 The on demand operating environment and the Enterprise Service Bus

The on demand operating environment is structured in the following three architectural layers:

- ▶ Infrastructure Services
- ▶ Integration Services
- ▶ Enterprise Service Bus (ESB)

You can see the role of the ESB in Figure 1-5.

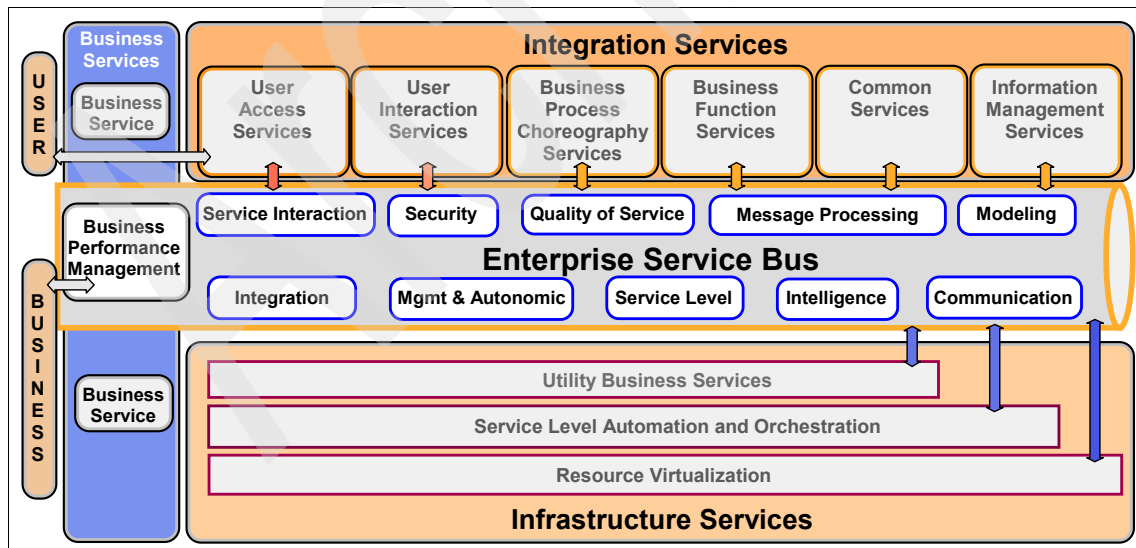


Figure 1-5 On demand operating environment based on SOA

Infrastructure services

The services provided in this architectural layer serve as a fundamental basis for the other two layers. The services in the infrastructure category are subdivided into the following:

- ▶ Utility business services
Includes functions such as billing, metering, rating, peering, and settlement.
- ▶ Service level automation and orchestration
Provides services that facilitate translation into reality of quality of service policy declarations associated with the business services.
- ▶ Resource virtualization
Provides instrumentation of server, storage, network, structured and unstructured information content, to enable management and virtualization of those resources under the control of on demand operating environment resource managers.

Typically, these services are built by middleware providers or Independent Software Vendors (ISVs) and held by the technical support team.

Integration services

The programming model for on-demand business services is based on building component (service) assembly. This means that services exposed in the integration services layer could be used by on-demand application builders to create new business services. The list of integration services includes the following categories:

- ▶ User access services
This services handles the adaptation of systems to specific devices (from desktop to pervasive devices), depending on their properties (display size, memory, processor, and so on).
- ▶ User interaction services
This type of service manages direct interactions with people involved in business processes.
- ▶ Business process choreography services
These services handle business logic represented in process flows, state machines, or business rules in order to describe one service in term of the interaction of others.

- ▶ **Business function services**

Services in this category provide business logic represented only as atomic operations that are required by the overall system. This includes adapters for packaged software or custom applications and the development of new services not implemented in the system.

- ▶ **Common services**

Common services implement functions useful for many business services. Among these functions are services implementing personalization, reporting, user access or user interaction, and so on.

- ▶ **Information management services**

The services in this category help integrate information and the respective meta data stored in different data sources in order to access it, analyze it and transform it.

In contrast to infrastructure services, these services are built by on-demand infrastructure and application builders, and held by the software support team, developers, technical staff, and business analyst.

For more information

Visit these Web sites for further information about the products in this section.

- ▶ For more information about SOA, refer to the IBM Service-Oriented Architecture (SOA) Web page:

<http://www-306.ibm.com/software/solutions/soa/>

- ▶ For more information about the IBM On Demand Operating Environment, visit the IBM On Demand Operating Environment Web page:

<http://www-3.ibm.com/software/info/openenvironment/>

- ▶ For more information about Web Services, visit:

<http://www-306.ibm.com/software/solutions/soa/>

1.3 WebSphere Process Server

WebSphere Process Server provides the runtime engine for artifacts produced in a business-driven development process. Figure 1-6 on page 18 shows how WebSphere Process Server fits into a business-driven development process and the surrounding tools that complement it.

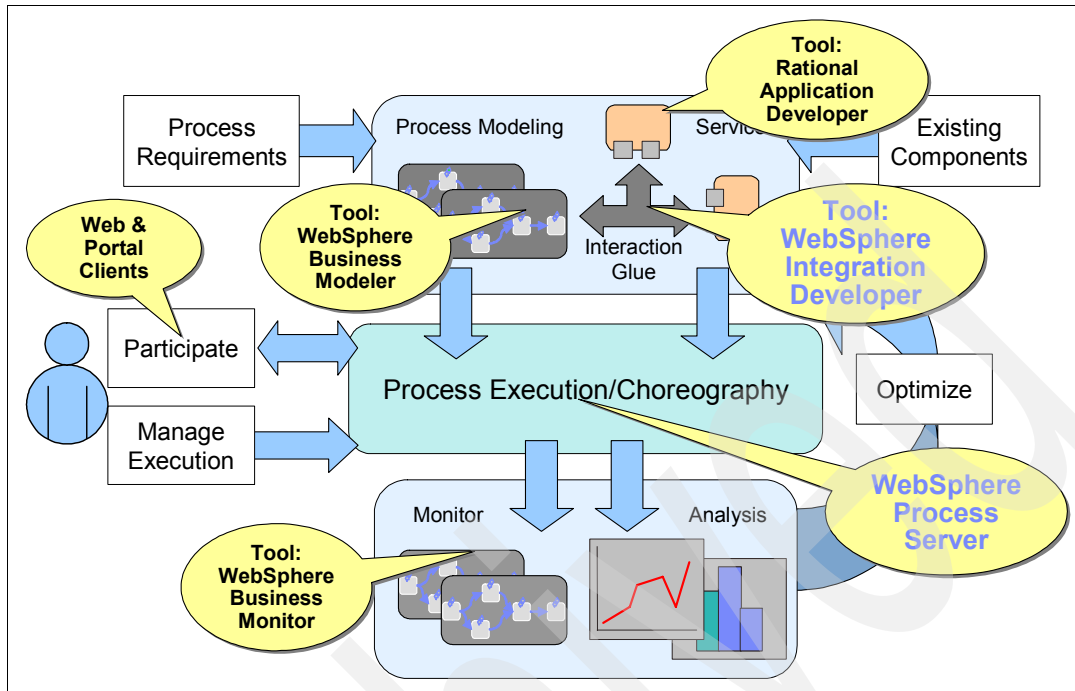


Figure 1-6 Business-driven development process and related tools

The next sections provide an overview of WebSphere Process Server in two ways:

- ▶ From the perspective of an architectural model, describing all the logical components of the product
- ▶ From the perspective of a programming model, where the main concern is to describe the way the different SCA artifacts can interact between them

In both perspectives, you can see easily the adapter technology at work.

1.3.1 Architectural model

The architectural model of WebSphere Process Server is shown in Figure 1-7.

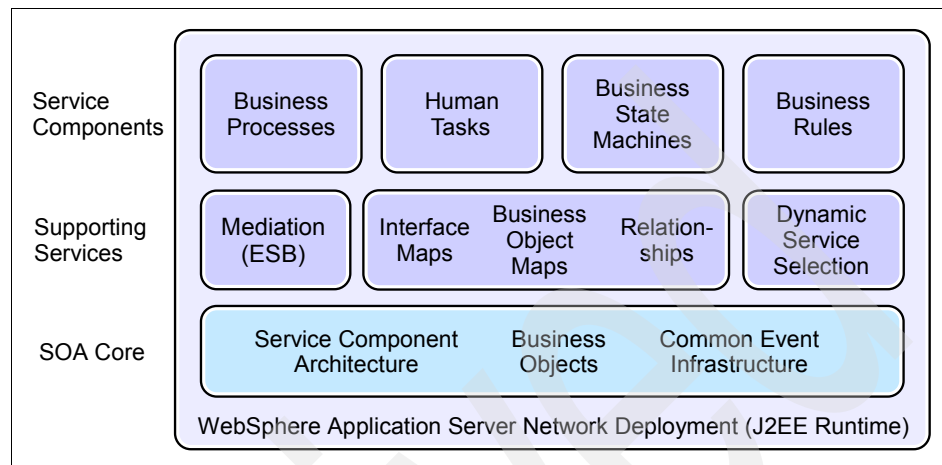


Figure 1-7 Architectural model of WebSphere Process Server

Technically, WebSphere Process Server is mounted on top of WebSphere Application Server, a very strong J2EE application server. In this way, WebSphere Process Server can take advantage of all the infrastructure services, for example transaction management, workload management, security, clustering, failover, and scalability. The J2EE server also includes a built-in messaging provider which can be configured to connect to an existing WebSphere MQ network.

Also in the infrastructure layer is the Common Event Infrastructure (CEI). This infrastructure is an implementation of a consistent approach for the creation, transmission, persistence, and distribution of events, and is based in the Common Business Event (CBE), an event definition that IBM has proposed to OASIS to be standardized.

On top of this infrastructure, WebSphere Process Server implements a layer called the SOA Core. This layer includes the fundamental concepts needed for an optimal integration process. This layer also unifies the invocation model (through Service Component Architecture (SCA) and the data model (through Service Data Objects (SDO) standard. SCA bindings describe the physical description of the components. Services can be accessed as Plain Old Java Objects (POJOs), Enterprise JavaBeans™ (EJBs), Web Services, JMS, messages, and adapters.

On top of the SOA Core layer, WebSphere Process Server implements several components and services that can be used in the development of an integration solution. These components are:

- ▶ *Business processes* representing a business process in terms of activities arranged in a workflow
- ▶ *Human tasks* modeling activities that require a human intervention inside a process
- ▶ *Business state machines* modeling event-based processes and defined in terms of states and transitions
- ▶ *Business rules* implementing business logic in a declarative way

In addition, there is a supporting services layer whose main function is to provide transformation and adaptation services for all the instances of the Service Components layer. These services are:

- ▶ *Interface maps* are devoted to matching semantically equal but syntactically different interfaces.
- ▶ *Business object maps* are used to transform one business object to another.
- ▶ *Relationships* establish different relations between business objects managed by different back end systems.
- ▶ *Dynamic service selectors* are components that allow the dynamic selection and invocation of the implementation of one interface.
- ▶ *Adapters* are components with a primary function to encapsulate the complexity of earlier systems and can be exposed as SCA components.
- ▶ *Mediation* is a component that can act on messages flowing between the requestor and the service.

1.3.2 Programming model

Figure 1-8 on page 21 shows some of the technologies that can be used in an integration project. This variety in technology can make the process of integration development difficult, and can create great expense during the development and deployment stages.

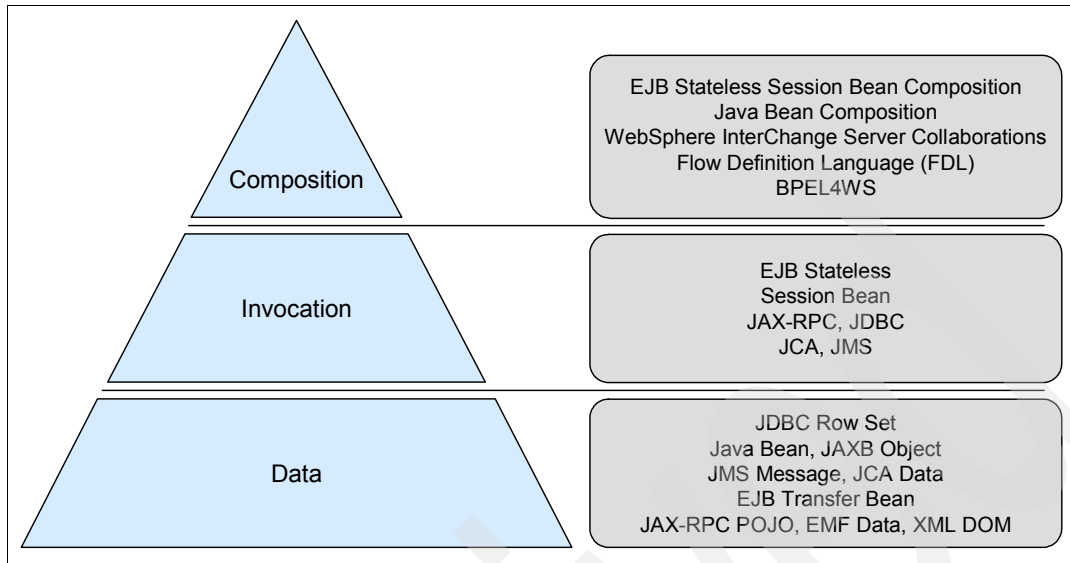


Figure 1-8 Programming model: today

IBM has simplified this choice in technology and exposes a new model of programming. This simplified model is based on three needs:

- ▶ The need to represent data
- ▶ The need for invoking services
- ▶ The need to compose services

This model is depicted in the Figure 1-9.

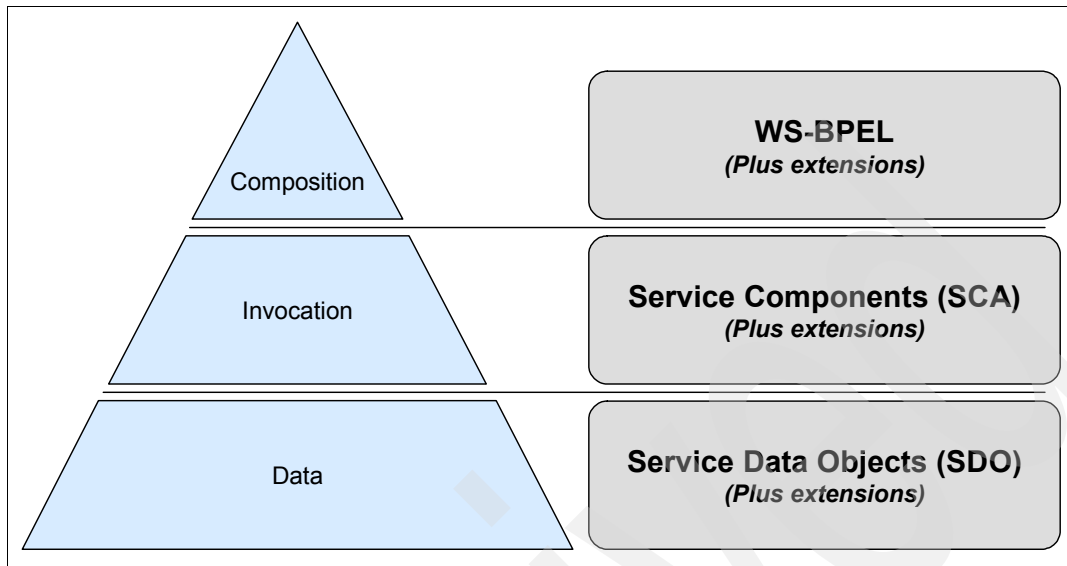


Figure 1-9 Programming model: simplification

For managing data, the Service Data Objects (SDO) standard together with some extensions are the basis for the realization of business objects. Business objects are the key to provide a universal means of describing data and accessing data. The extensions named are used to implement additional functionality like metadata support, context information and change history management.

At the invocation level, WebSphere Process Server supports Service Component Architecture (SCA). SCA describes all the integration artifacts (processes, business rules, human tasks and so on) as service components and permits grouping them together into modules. In this way, it is possible to model and deploy a complete integration solution.

At the composition level, WebSphere Process Server uses Business Process Execution Language for Web Services (BPEL4WS) or Web Services Business Process Execution Language (WS-BPEL) to describe and orchestrate the interaction between service components.

1.3.3 Service Component Architecture

Service Component Architecture (SCA) is a service-oriented component model for business services that publish or operate on business data. The advantage of SCA is that it provides a single abstraction for service types that can be

expressed already as session beans, Web Services, Java classes, BPEL flows, and so on. The basic elements of SCA are the service component and the service module.

Service component

A *service component* is an artifact integrated by the following elements:

- ▶ interface
- ▶ implementation
- ▶ reference

Figure 1-10 shows the interaction between these elements.

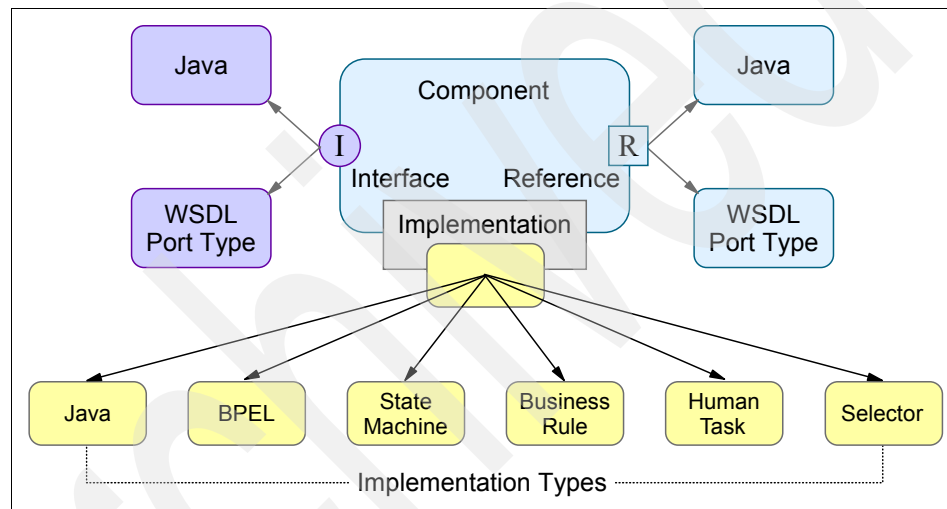


Figure 1-10 Service component: overview

An interface provides the input and output of a component. It is created independent of the internal implementation of the component. An interface is defined by a Java interface or WSDL port type. All components support WSDL type interfaces, but only Java components support Java interfaces. A component can be called synchronously or asynchronously independently of the implementation.

The implementation of the component can be developed as any of the following types:

- ▶ Java
- ▶ BPEL
- ▶ State machines
- ▶ Business rules

- ▶ Human task
- ▶ Selectors
- ▶ Adapters

SCA and non-SCA services can use other service components in their implementation. They can declare these dependencies as soft-links called *references*. If both components are inside the same module, the reference is in-line, but to use a service component defined outside the current module, you need to wire the component to that module service.

Imports and exports define a module's external interfaces or access points. Imports identify services outside of a module, so they can be called from within the module. Exports allow components to provide their services to external clients.

Applications that are not defined as SCA components, Java Server Pages (JSPs) for example, can still invoke SCA components; they do so through the use of standalone references.

Figure 1-11 depicts the types of existing references.

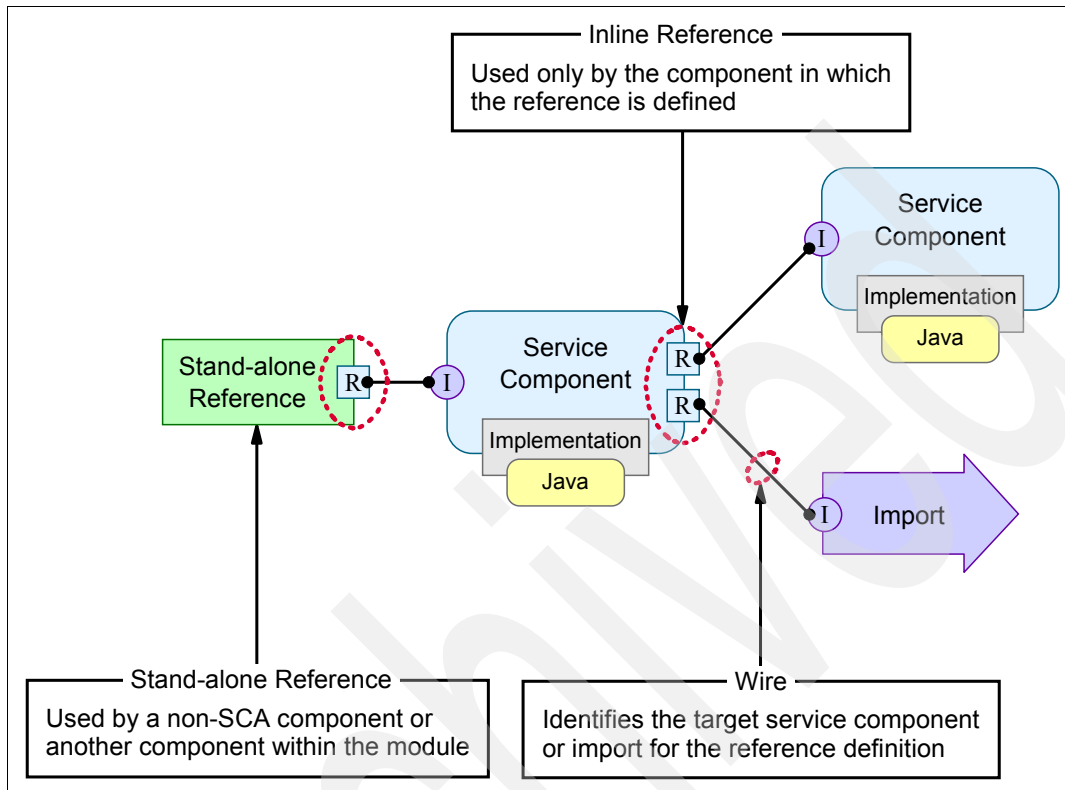


Figure 1-11 Service component and references

The details of a service component are stored in an XML file based on a new proposed standard: Service Component Definition Language (SCDL).

Service module

A service module is a logical group of components. The implementation of components that belongs to the module may reside within the module. If a component must invoke a component from another module, it must be done using an import.

Figure 1-12 shows the overview of a module.

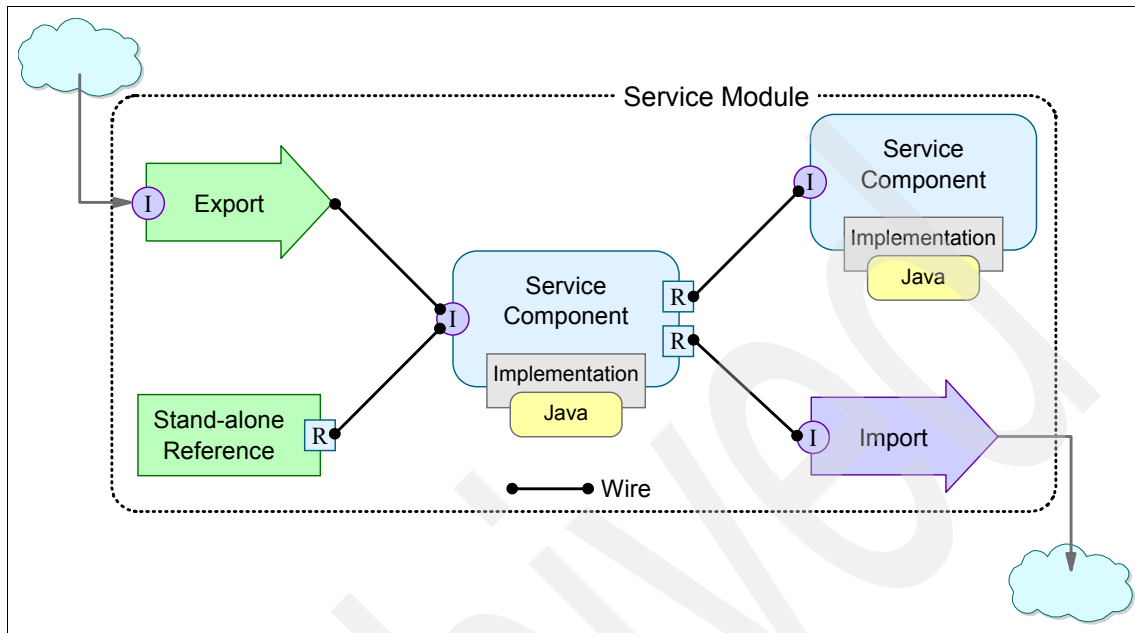


Figure 1-12 Service module: overview

1.3.4 Handling data

The basis for managing data in the programming model of WebSphere Process Server is the concept of a business object. Basically, a *business object* is a fundamental data structure used to represent business data. The concept of business objects in WebSphere Process Server is similar to the concept used in the IBM Interchange Server server product. The difference is that the new business object definition is based on the Service Data Object v1.0 technology standard and provides functionality above what is found in SDO. SDO is a proposed standard published jointly by IBM and BEA as JSR 235.

As the concepts of SDO and business object are very important in the development of adapters, explained in more detail in Chapter 3, "Service Data Objects" on page 53.

1.3.5 Composition: Business Process Execution Language

Although the last element in the programming model of WebSphere Process Server is not completely related to the development of adapters, it is an important concept in understanding how an adapter is related to other SCA

components. This element is called composition. *Composition* is the orchestration of multiple service components in a special type of implementation component: the business process service component. The main objective of this component is to provide a way to integrate business-to-business and enterprise applications by invoking and interacting with individual services and components modeled as activities. It describes the logical order and specific rules and conditions that must be satisfied in order to achieve a specific business goal.

A process component is described in Business Process Execution Language (BPEL), sometimes referred as Business Process Execution Language for Web Services (BPEL4WS) or Web Services Business Process Execution Language (WS-BPEL). WS-BPEL defines a model and a grammar for describing the behavior of a business process based on interactions within the process and its interactions with external partners. A *partner* can be any entity which provides a service, consumes a service, or both.

Figure 1-13 shows an example of a BPEL process service component.

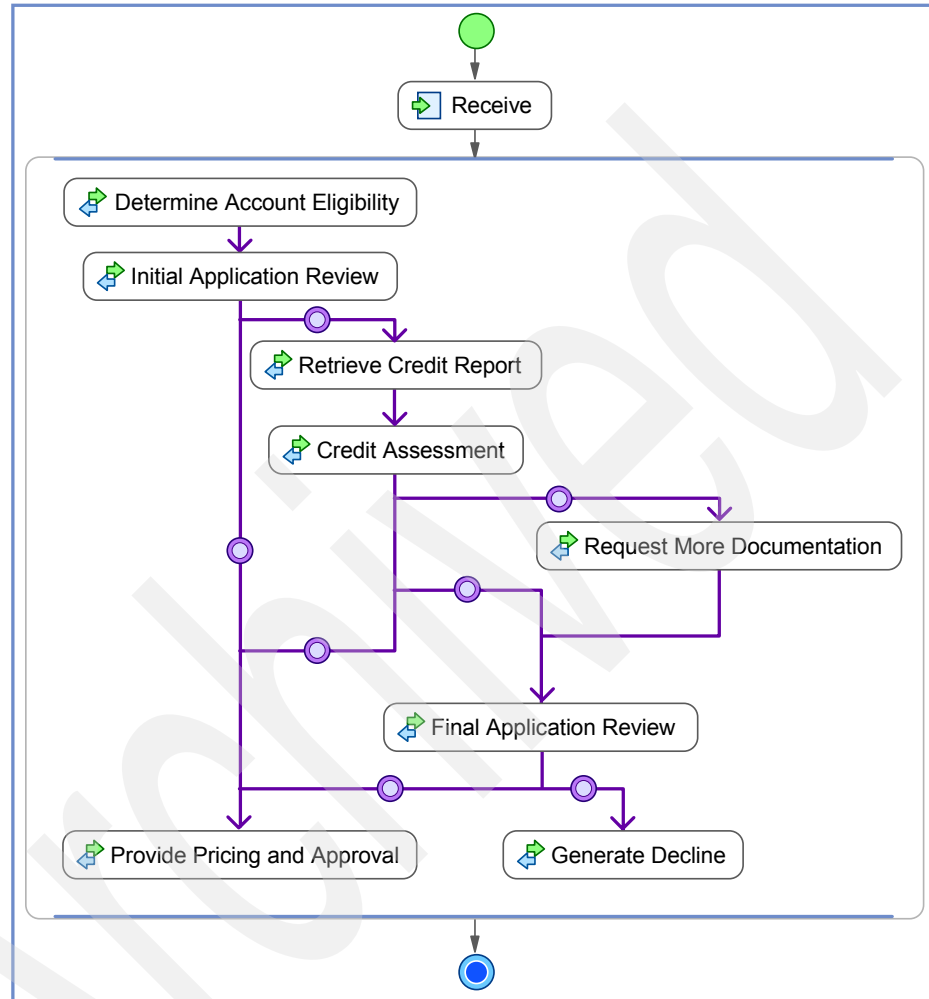


Figure 1-13 BPEL Process example

WS-BPEL is founded on an XML notation for specifying business process behavior based on Web Services. This standard is being directed through OASIS.

1.3.6 Other service implementation types

In addition to BPEL business processes, the current integration requirements of organizations need additional artifact types (such as incorporating human task

activities, integrating with Java applications, modeling business rules, or implementing event logic oriented orchestration) in order to build a successful solution.

WebSphere Process Server provides the following service implementation types:

- ▶ The *Java object* facilitates the implementation of a service as a Java class.
- ▶ The *Business state machine* is a model which describes a business process by focusing on the events that cause a transition from one state to another. The business state machine is used in applications where logic can be modeled easily as state machines or in applications that are activity oriented.
- ▶ The *Human task manager* provides a way to involve human interactions within a business process. There are three main types of human tasks:
 - *Participating* task service component creates a work item for human interaction.
 - *Originating* task requires a human interaction to invoke a service.
 - *Human* task requires human interaction to invoke a service creating a work item for another human to complete.
- ▶ *Business rules* represent a common way to implement and enforce business policies such as marketing heuristics, marketing principles, and government regulations in a declarative and independently manner.

1.3.7 For more information

For more information about WebSphere Process Server, refer to *Technical Overview of WebSphere Process Server and WebSphere Integration Developer*, REDP-4041.

For more information about the BPEL standard, refer to:

http://www.oasisopen.org/committees/tc_home.php?wg_abbrev=wsbpe1

1.4 WebSphere Integration Developer and Rational Application Developer

IBM offers products to help organizations in the process of building and integrating SCA components into a new solution. These products are geared toward specific roles in the development process.

The Figure 1-14 shows the relation between development roles and IBM development tools in a business driven development environment.

Business Driven Development - Roles and Tools

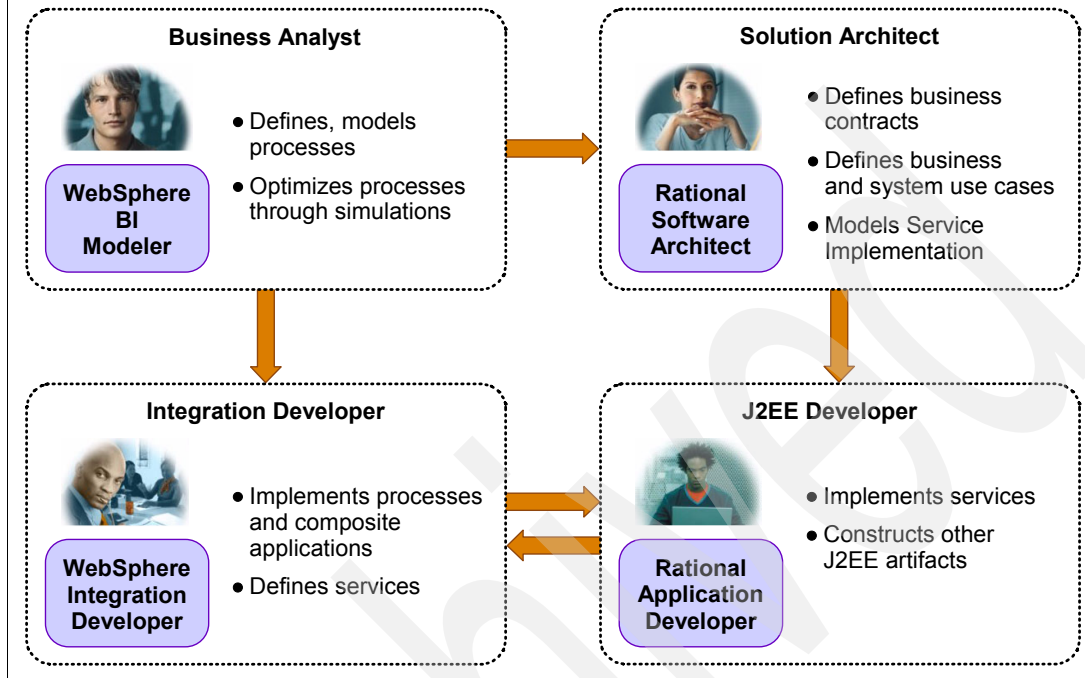


Figure 1-14 Business Driven Development: Roles and Tools

In this book we focus on the integration and J2EE developer roles and the corresponding development tools.

1.4.1 Integration developer

The integration developer focuses on building service oriented solutions. He has a basic understanding of business modeling and is familiar with basic programming concepts (loops, conditions, string manipulation). However, the integration developer lacks the programming skills required to develop a component from scratch. The integration developer works with business analysts as he builds the implementation for business processes.

WebSphere Integration Developer is a visual tool used by the integration developer to implement business process logic in terms of SCA modules and components, assemble these artifacts to produce a complete solution, and synchronize the artifacts with the business analysis models.

1.4.2 Software (J2EE) developer

The J2EE developer has high-level skills in the J2EE development platform and a basic understanding of SOA, process choreography, workflow, WSDL and BPEL concepts. He is responsible for implementing application-specific business logic for integrated solutions and exposes application logic as a service through the development or use of adapters.

Rational® Application Developer (RAD) is the development tool used to create J2EE and Web Services artifacts. These artifacts are then used by the integration developer to assemble solutions and to automate the low-level details of the J2EE programming model.

1.5 Adapters in process integration

Business enterprise uses Enterprise Information Systems (EIS) to provide the information infrastructure for the business. Adapters provide a mechanism for the integration of existing EIS infrastructure with process integration applications. This is a service-oriented approach to EIS integration. Adapters abstract or expose the low-level EIS functions or events in the form of business objects. This enables service components running WebSphere Process Server to interact with EIS by passing business objects. It saves the service component having to deal with the low APIs and events of the EIS. Adapters provide a consistent framework to interact with EIS.

Figure 1-15 shows EIS integration using adapters.



Figure 1-15 EIS integration using adapters

Figure 1-16 shows adapter's role in WebSphere Process Server.

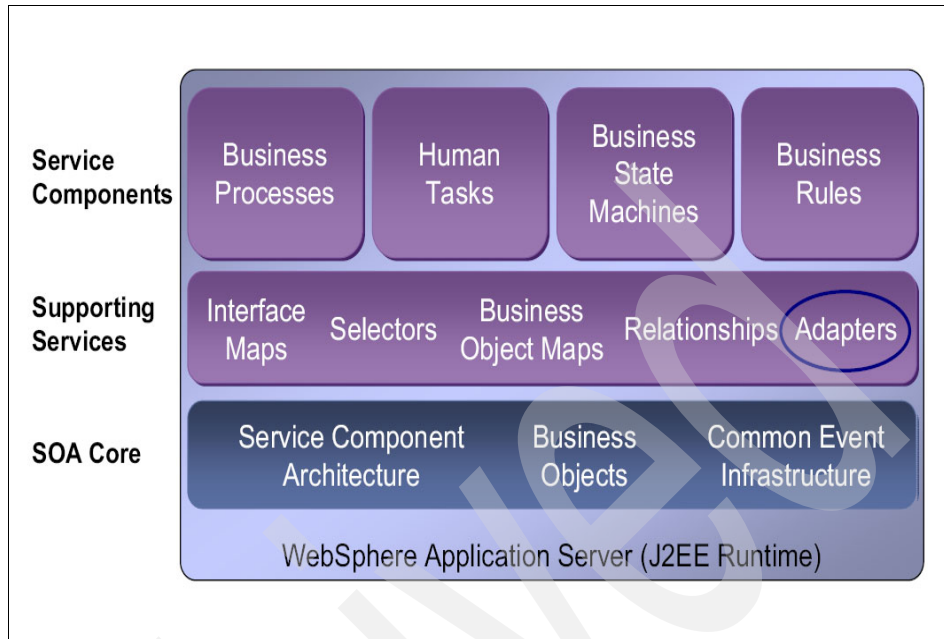


Figure 1-16 The role of the adapter in WebSphere Process Server

Adapter basics

This chapter introduces adapters and how they are used. It discusses Java 2 Enterprise Edition (J2EE) Connector Architecture (JCA) first. Then, it introduces the IBM WebSphere Adapters.

This chapter includes the following topics:

- ▶ Introduction to adapters
- ▶ Overview of JCA resource adapters
- ▶ IBM WebSphere Adapters
- ▶ WebSphere Business Integration Adapters
- ▶ IBM WebSphere Adapter versus JCA resource adapter

2.1 Introduction to adapters

An *adapter* is a device that is used to connect two items of dissimilar shape, size, or orientation to each other. Consider a businessman traveling around the world with his IBM ThinkPad. If he wants to connect his Thinkpad (bought in North America) into an electrical outlet in Great Britain, Australia, or South Africa, the Thinkpad power cord would not fit the outlet. He would need different types of plugs to connect to electrical outlets in each of these countries. To solve this problem, the business man buys adapters to work with each country's electric outlets and then attaches his Thinkpad to the adapter. One end of the adapter can receive the North American plug from the Thinkpad. The other end can attach to another country's electrical outlet. In other words, the adapter converts a country-specific electric outlet interface to a North American open-standard outlet interface. This means any client appliance from North America can connect to the other country's electric outlet using an adapter (assuming that the device can handle different voltages from different countries).

This simple example in Figure 2-1 shows the usefulness of adapters and the importance of adapter support for open standards.

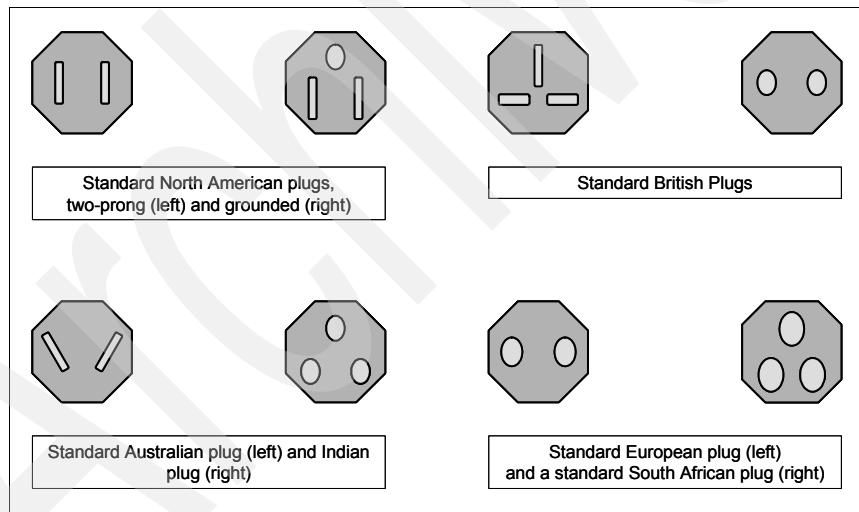


Figure 2-1 Electric plugs of other countries

Software adapters

The simple idea used in the electrical outlet example can also be applied to software as well. We can have software adapters. There is a software design pattern called the Adapter pattern illustrated in Figure 2-2.

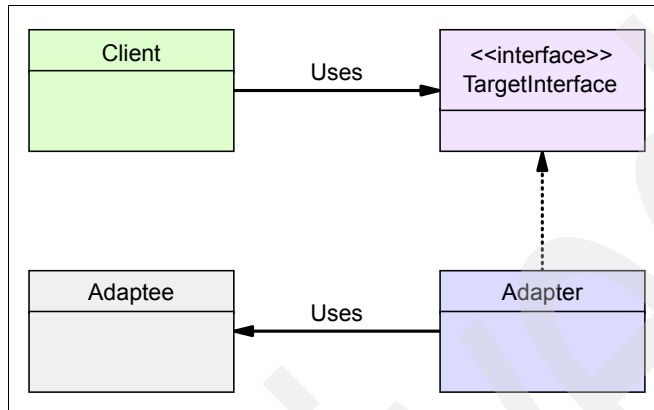


Figure 2-2 Adapter pattern

A software adapter, also called a *wrapper*, converts existing interfaces of a component to new interfaces so the client of this component can connect to it and invoke operations. The idea is that we want components to work with each other even when they have not have been designed to do so originally.

This book focuses on software adapters that comply with the J2EE Connector Architecture (JCA) Specification. For more information about the JCA specification, see:

<http://java.sun.com/j2ee/connector/index.jsp>

2.2 Overview of JCA resource adapters

In Chapter 1, "Introduction and adapter overview" on page 3, we describe the IBM vision of On Demand Business. *Integration* is one of the key technology attributes that provides the basis for the On Demand Business message. Integration can occur between people, processes, applications, systems and data. J2EE Connector Architecture (JCA) adapters play a key role in the integration of applications and data using open standards. This section presents an overview of JCA compliant resource adapters.

2.2.1 Motivation for JCA adapters

As of the early 1970s, organizations have employed Enterprise Information Systems (EIS) to provide the back-end infrastructure for businesses. The EIS could be an enterprise resource planning (ERP) system, a supply chain management (SCM), or a transaction processing monitor (TP monitor). Organizations have invested enormous resources on these off-the-shelf EIS and custom built applications. As more organizations move to provide Web-based solutions, it is important that Web-based applications have a way to integrate with these EIS applications to leverage existing enterprise services, business logic and data. It is also important for organizations to be able to integrate the EIS applications with each other. Adapters, in conjunction with IBM WebSphere Process Server, provide the necessary infrastructure to facilitate this integration.

Prior to the release of the J2EE Connector Architecture (JCA) specification in 2001, there was no standard, consistent way to integrate an EIS with a J2EE application. If an EIS vendor supported connectivity to an application server, it had its own method of connecting and often required custom code.

Figure 2-3 illustrates this complexity.

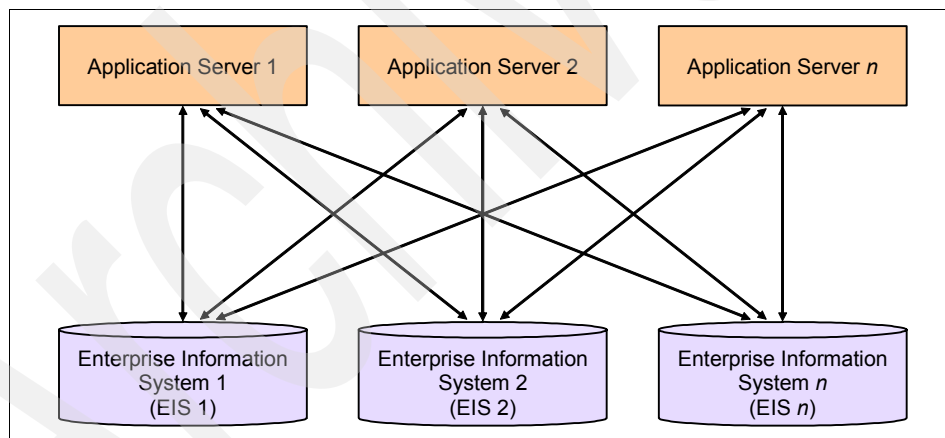


Figure 2-3 Complexities of point to point integration

JCA standardizes the way J2EE application components, J2EE compliant application servers and EIS resources interact with each other. To achieve this interaction, the JCA specification defines a set of system contracts for both the application server and the EIS. The EIS side of the system contract is implemented by a resource adapter. Very much like a JDBC™ driver facilitates connects to a database, a resource adapter is a system level software driver that allows Java applications to connect to a specific EIS. The resource adapter plugs into the application server to provide connectivity between the application

component, application server and EIS. Providing a JCA compliant resource adapter for an EIS ensures that the EIS can connect to and interact with any JCA compliant application server. Figure 2-4 illustrates how a resource adapter for an EIS can be reused in multiple JCA compliant application servers.

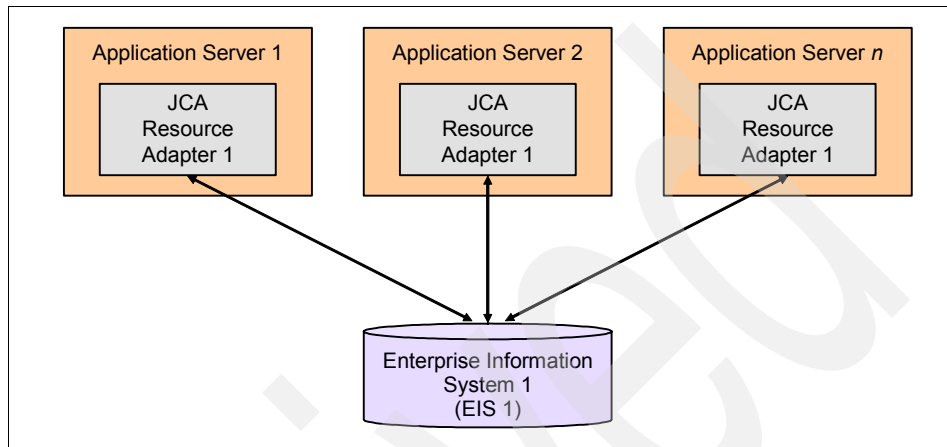


Figure 2-4 Application servers using resource adapter to interact with EIS 1

Figure 2-5 illustrates how one application server can connect to multiple EIS resources, each having its own resource adapter. Figure 2-4 in conjunction with Figure 2-5 illustrates that for M application servers and N EIS resources, integration has been simplified from M x N complexity to M +N complexity.

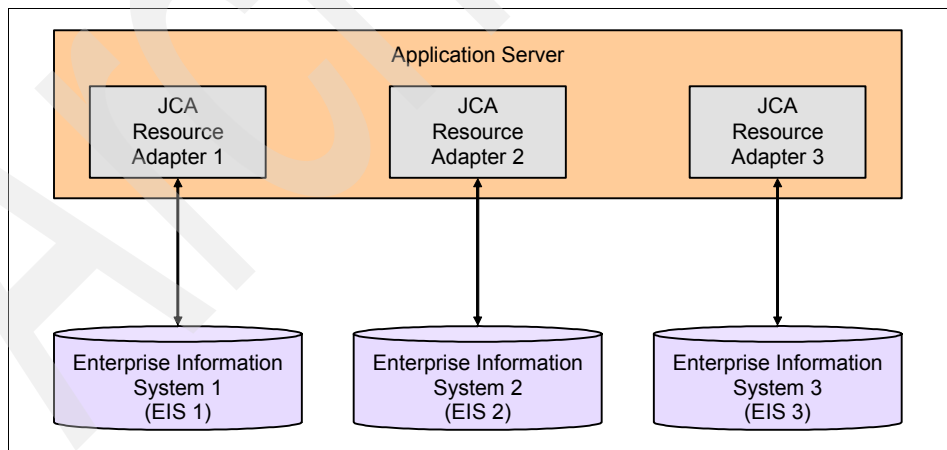


Figure 2-5 Application server with multiple EISs using a resource adapter

2.2.2 J2EE Connector Architecture overview

The JCA specification defines a standard for enabling connectivity and interaction between application server and EIS. Resource adapters and application servers implement the contract defined in the JCA specification. Resource adapters run in the context of the application server and enable J2EE application components to interact with the EIS using a common client interface. JCA compliant application servers can support any JCA compliant resource adapter and vice-versa. Figure 2-6 illustrates an application component connected to an EIS through the resource adapter.

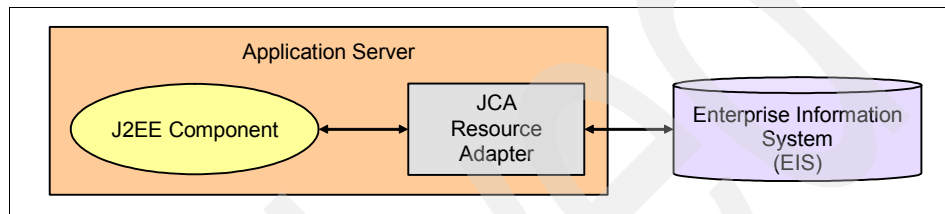


Figure 2-6 Resource adapter deployed to an application server

Note: In addition to supporting the access of EIS resources from application components running on an application server (managed environment), the JCA specification also supports access to EIS resources from Java applications and applets running outside of an application server (nonmanaged environment). Using a resource adapter in a managed environment is much easier to implement because the application server provides the life cycle, connection, transaction, and security management services. This book focuses on accessing EIS resources in a managed environment.

System contracts

System contracts define the connection and interaction between the application server and EIS. The application server side of the system contract is implemented by the JCA container while the EIS side of the system contract is implemented by a resource adapter for a specific EIS. The application server and the resource adapter connect and interact with each other using the system contracts.

Figure 2-7 on page 39 shows the integration between EIS, application server and application component. These components are bound together using JCA contracts.

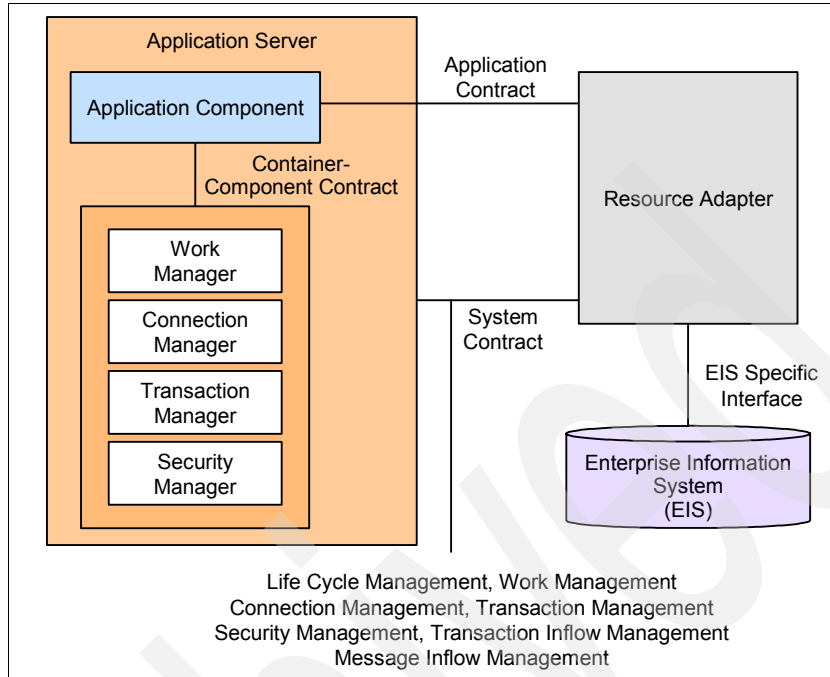


Figure 2-7 JCA system contract: application server and resource adapter

Note: The resource adapter resides in the application server. In Figure 2-7, it is shown outside of the application box to illustrate the system contracts.

Seven types of system contracts are defined in the JCA specification.

Life cycle management contract

The life cycle management contract allows the application server to manage the life cycle of the resource adapter. It provides a mechanism for the application server to start and shutdown an instance of the resource adapter. It also allows the resource adapter to be notified during undeployment and orderly shutdown of the application server.

The key interface that must be implemented by the resource adapter is the `ResourceAdapter` interface. The key methods are `start` and `stop`. The class name of this implementing class must be specified in the resource adapter deployment descriptor. During adapter deployment or application server startup, the application server calls the `start` method of the implementing class to bootstrap an instance of the resource adapter. When the application server is shutting down or the resource adapter is being undeployed, the application server stops

the application components that are using the resource adapter and then calls the stop method of the implementing class of ResourceAdapter interface. This allows the resource adapter to shutdown gracefully.

Work management contract

The work management contract allows the resource adapter to submit work to the application server for execution. The application server dispatches a thread to handle the work. This mechanism delegates the management of the thread to the application server. This contract is optional. The key interfaces are Work, ExecutionContext, and WorkListener.

Connection management contract

The connection management contract allows the application server to create a physical connection to the EIS. It also provides a mechanism for the application sever to manage connection pooling. The key interfaces are ConnectionFactory, Connection, ConnectionRequestInfo, ManagedConnectionFactory, ManagedConnection, and ManagedConnectionMetaData. ManagedConnectionFactory, ManagedConnection, and ManagedConnectionMetaData are implemented by the resource adapter.

Figure 2-8 shows the connection management and connection pooling mechanism specified in the connection management contract.

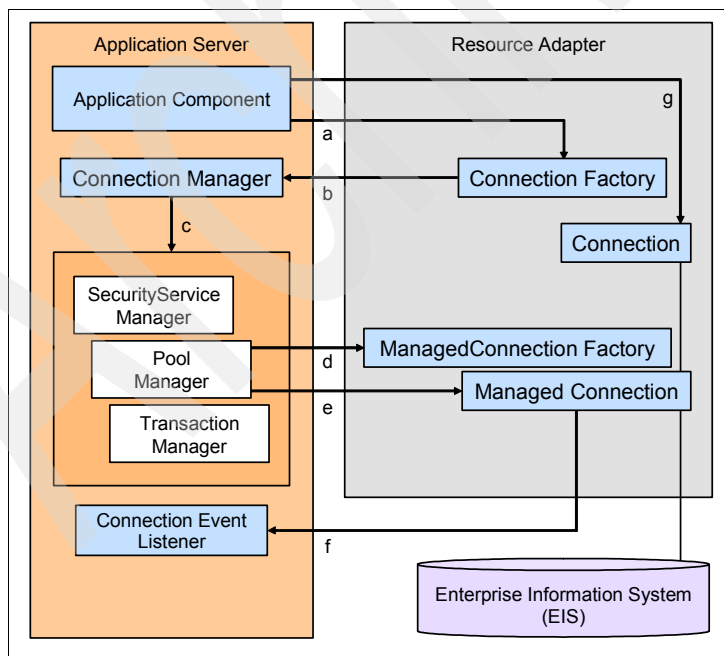


Figure 2-8 Adapter connection management architecture

Figure 2-8 on page 40 illustrates the steps taken when a J2EE application component obtains a connection to an EIS using a JCA-compliant application server and a JCA-compliant adapter:

1. The application component performs a lookup of the connection factory in the adapter from the Java Naming and Directory Interface™ (JNDI) name space.
2. The connection factory instance from the adapter delegates the connection request to the connection manager on the application server.
3. The connection manager instance performs a lookup in the connection pool to find an available connection.
4. If there is no available connection to the EIS, the pool manager uses the managed connection factory to create a new physical connection to the EIS (a managed connection instance). The pool manager then adds the new managed connection to the connection pool and returns it to the connection manager.
5. If the connection manager finds an available connection, it returns this managed connection instance to the connection manager.
6. In order to be notified of events in the managed connection, the application server registers a connection event listener with a managed connection.
7. The connection manager uses the managed connection instance to obtain a connection. This connection is the application handle to the physical connection to the EIS. This connection instance is returned to the application component for interaction with the EIS.

Transaction management contract

The transaction management contract provides transactional support allowing the EIS to participate in a transaction. Transactions can be managed by the application server's transaction manager with multiple EISs and other resources as participants. If only a single EIS is involved, the transaction can be managed internally by the EIS. The resource adapter can choose to support both types of transactions, local transactions only, or neither transaction type. The key interfaces are: LocalTransaction and XAResource. Refer to 4.12, "Transaction support" on page 102 for further detail.

Figure 2-9 shows the transaction management contract.

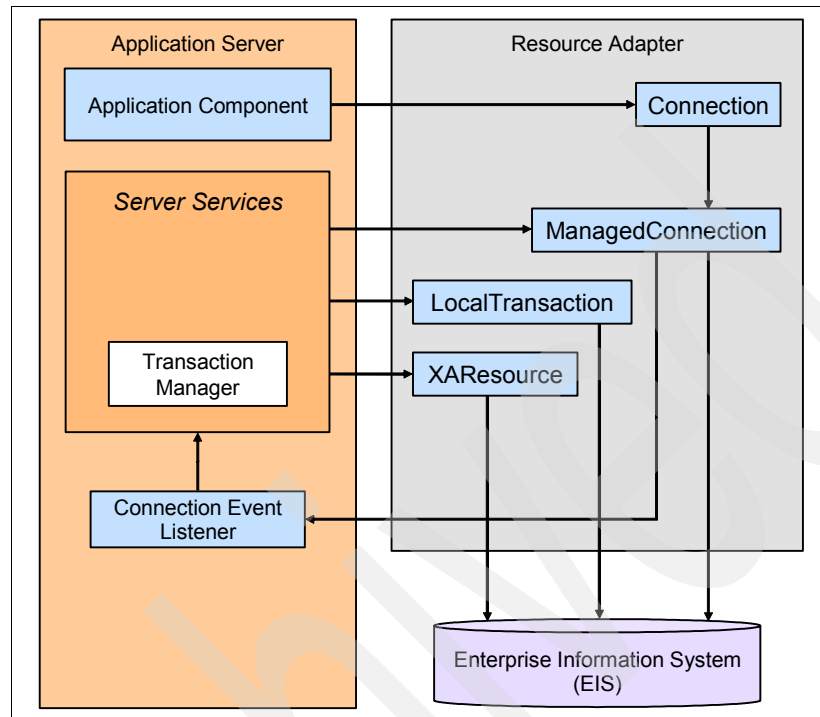


Figure 2-9 Transaction management

In order to support local transactions, the ManagedConnection interface provides a `getLocalTransaction` method. This method returns a LocalTransaction instance. The methods in the LocalTransaction interface are `begin`, `commit`, and `rollback`. The implementation of these methods cause the EIS to begin, commit, and rollback a transaction.

The ManagedConnection object notifies its registered ConnectionEventListener whenever an application component does a `begin`, `commit`, or `rollback` on a transaction. It calls the following corresponding methods in ConnectionEventListener:

- ▶ `localTransactionStarted`
- ▶ `localTransactionCommitted`
- ▶ `localTransactionRolledback`

To support a transaction across multiple EIS or other resources, the transaction manager uses the XAResource interface to perform two-phase commit. The ManagedConnection interface provides a `getXAResource` method. This method

returns a XAResource instance. The methods in the XAResource interface are commit, end, forget, prepare, recover, rollback and start.

For more information about XA transaction, see the JCA specification:

<http://java.sun.com/j2ee/connector/index.jsp>

Security contract

The security contract allows application components in an application server to access the EIS securely. The security contract is an extension of the connection management contract implemented by adding J2EE Java Authentication and Authorization Service (JAAS) into connection management interfaces.

Figure 2-10 shows the security management contract.

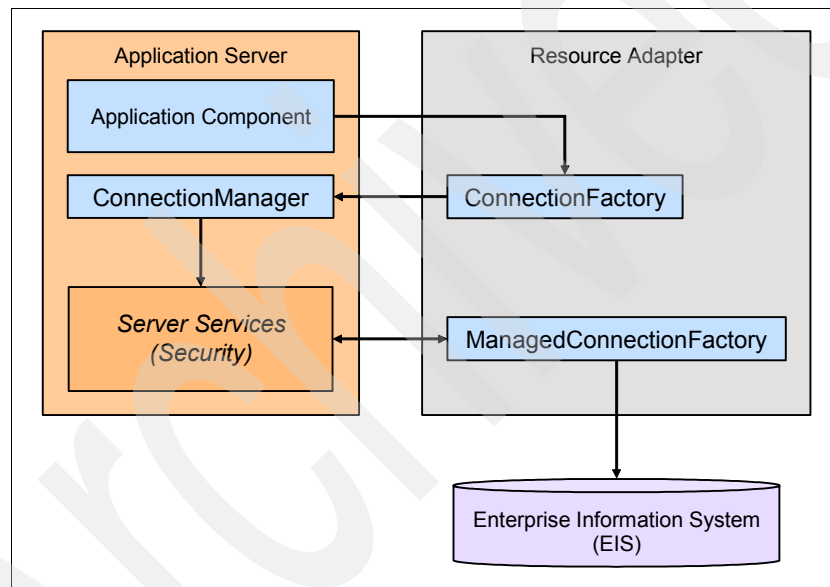


Figure 2-10 Security management

Security in the JCA specification use the JAAS Subject class to provide security information. When the application server calls `ManagedConnectionFactory.createManagedConnection` to get a `ManagedConnection` instance, it passes a `Subject` instance. The `Subject` instance contains the security information needed to connect to an EIS.

JCA defines two types of credentials that can be used as a security principal:

- ▶ **GSSCredential**

GSSCredential is a generic credential that can be used to represent a specific security system, such as Kerberos.

- ▶ **PasswordCredential**

PasswordCredential contains user name and password information.

The resource adapter deployment descriptor must specify the type of security supported along with the authentication mechanism used. It also must list the supporting credential interface and whether reauthentication is supported.

Transaction inflow contract

The transaction inflow contract allows the EIS to propagate a transaction through the resource adapter to the application server. To preserve the ACID property of the incoming transaction, this contract enables the resource adapter to flow-in transaction completion and failure recovery calls from the EIS to the application server.

Message inflow contract

The message inflow contract allows the resource adapter to pass synchronous or asynchronous inbound messages to message endpoints on the application server. This contract does not require a specific messaging style, messaging semantics, or messaging infrastructure. The key interfaces are ResourceAdapter and ActivationSpec.

Common Client Interface

The Common Client Interface (CCI) is a standard API that allows application components and Enterprise Application Integration (EAI) frameworks to interact with the resource adapter. It provides a standard way for application components to invoke functions on an EIS and get the returned results. IBM WebSphere Adapters use CCI for outbound communication with an EIS. CCI has five types of interfaces:

- ▶ **Connection related interfaces**

Application components connect to the resource adapter through these interfaces:

- javax.resource.cci.ConnectionFactory
- javax.resource.cci.Connection
- javax.resource.cci.ConnectionSpec
- javax.resource.cci.LocalTransaction

- ▶ **Interaction related interfaces**

Application components interact with the resource adapter through these interfaces. For example, the application component can invoke an EIS function by calling the execute method of the InteractionImpl class.

- javax.resource.cci.Interaction
- javax.resource.cci.InteractionSpec

► Data representation related classes

These classes are used to represent data:

- javax.resource.cci.Record
- javax.resource.cci.MappedRecord
- javax.resource.cci.IndexRecord
- javax.resource.cci.RecordFactory
- javax.resource.cci.Streamable
- javax.resource.cci.ResultSet
- javax.sql.ResultSetMetaData

Note that WebSphere Adapters use Service Data Objects (SDO) to represent data instead of these CCI interfaces.

► Metadata related interfaces

The implementation classes of these interfaces provide meta-information on the EIS and connection to the EIS. For example, ConnectionMetaDataImpl provides information about EIS name, EIS version, and user name. ResourceAdapterMetaDataImpl provides information about adapter name, adapter version, adapter vendor, adapter description, and supported JCA specification version.

- javax.resource.cci.ConnectionMetaData
- javax.resource.cci.ResourceAdapterMetaData
- javax.resource.cci.ResultSetInfo

► Exceptions and warnings

- javax.resource.cci.ResourceException
- javax.resource.cci.ResourceWarning

Service Provider Interface (SPI)

The resource adapter communicates with its hosting application server through the Service Provider Interface. It provides a common infrastructure programming model for resource adapters. This allows the adapters to have a standard view of their environment. The Service Provider Interface has the following:

► Adapter Startup Interfaces

- javax.resource.spi.BootstrapContext
- javax.resource.spi.ResourceAdapter
- javax.resource.spi.ResourceAdapterAssociation

- Event Management Interfaces
- javax.resource.spi.endpoint.MessageEndpoint
- javax.resource.spi.endpoint.MessageEndpointFactory
- ▶ Connection Management Interfaces
 - javax.resource.spi.ActivationSpec
 - javax.resource.spi.ConnectionEvent
 - javax.resource.spi.ConnectionEventListener
 - javax.resource.spi.ConnectionManager
 - javax.resource.spi.ConnectionRequestInfo
 - javax.resource.spi.DissociatableManagedConnection
 - javax.resource.spi.LazyAssociatableConnectionManager
 - javax.resource.spi.LazyEnlistableConnectionManager
 - javax.resource.spi.LazyEnlistableManagedConnection
 - javax.resource.spi.ManagedConnection
 - javax.resource.spi.ManagedConnectionFactory
 - javax.resource.spi.ManagedConnectionMetaData
 - javax.resource.spi.ValidatingManagedConnectionFactory
- ▶ Security Interfaces
 - javax.resource.spi.security.GenericCredential
 - javax.resource.spi.security.PasswordCredential
- ▶ Work Interfaces
 - javax.resource.spi.work.ExecutionContext
 - javax.resource.spi.work.Work
 - javax.resource.spi.work.WorkAdapter
 - javax.resource.spi.work.WorkEvent
 - javax.resource.spi.work.WorkListener
 - javax.resource.spi.work.WorkManager
- ▶ Transaction Interfaces
 - javax.resource.spi.LocalTransaction
 - javax.resource.spi.XATerminator
- ▶ SPI Exceptions
 - javax.resource.spi.ApplicationServerInternalException
 - javax.resource.spi.CommException
 - javax.resource.spi.EISSystemException
 - javax.resource.spi.IllegalStateException
 - javax.resource.spi.InvalidPropertyException
 - javax.resource.spi.LocalTransactionException
 - javax.resource.spi.ResourceAdapterInternalException

- javax.resource.spi.ResourceAllocationException
- javax.resource.spi.SecurityException
- javax.resource.spi.SharingViolationException
- javax.resource.spi.UnavailableException
- javax.resource.spi.work.WorkCompletedException
- javax.resource.spi.work.WorkException
- javax.resource.spi.work.WorkRejectedException

2.2.3 Resource adapter packaging

Resource adapter modules contain the necessary files to provide connectivity and interaction between application component, application server and EIS. Typically, the resource adapter contain the following files:

- ▶ Java classes and interfaces that implement the EIS side of JCA specification (the resource adapter)

These files are packaged in a JAR file.

- ▶ Any dependent libraries required by the resource adapter
- ▶ Any platform-specific dependent EIS native libraries
- ▶ The deployment descriptor

The deployment descriptor lists the fully qualified names for the required resource adapter implementation classes and interfaces. This helps the application server to load and use the appropriate classes at runtime. Transaction and security information are also express in the deployment descriptor:

- ManagedConnectionFactory implementation class
- ConnectorFactory interface
- ConnectionFactroy implementation class
- Connector interface
- Connection implementation class

2.3 IBM WebSphere Adapters

The IBM WebSphere Adapter portfolio is a new generation of adapters based on the Java 2 Platform, Enterprise Edition (J2EE) standard. An IBM WebSphere Adapter implements the JCA Specification version 1.5. Also known as resource adapters or JCA adapters, WebSphere Adapters enable managed, bidirectional connectivity and data exchange between EIS resources and J2EE components supported by WebSphere Process Server. See Figure 2-11 on page 48.

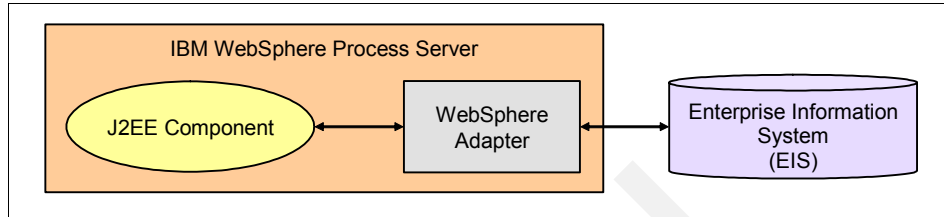


Figure 2-11 IBM WebSphere Adapter

WebSphere Adapters use Service Data Objects (SDO) for representing data objects.

Each WebSphere Adapter consists of the following:

- Foundation classes

These implement a generic set of contracts that WebSphere Process Server uses to manage interactions between J2EE applications and all WebSphere Adapters. These contracts define the SPI.

When developing custom adapters, we strongly recommend developing IBM WebSphere resource adapters (as oppose to J2EE resource adapters). This is because you can extend and utilize services and system contracts already implemented in the adapter foundation classes. For example, you can utilize the command pattern implemented in the adapter foundation classes to process hierarchical business objects. This saves a lot of development effort when creating custom adapters.

- EIS-specific subclasses of the foundation classes

These EIS-specific subclasses provide additional support for the SPI contract as well as the CCI. For example, an EIS-specific ConnectionFactory implementation enables components in the WebSphere Process Server to receive a handle to the underlying EIS application.

When developing custom WebSphere resource adapters, these EIS-specific subclasses need to be implemented.

- Enterprise Metadata Discovery component

The Enterprise Metadata Discovery specification introduces a new metadata discovery and import model for resource adapters and the Enterprise Application Integration (EAI) tools framework. The specification defines a common API that resource adapters use to expose services and business objects to tools for the generation of JCA-based applications.

IBM WebSphere Adapters provide an Enterprise Metadata Discovery component that complies with the specification, except IBM CICS® ECI Resource Adapter 6.0.1 and IBM IMS™ Connector for Java 9.1.0.2.

WebSphere Integration Developer provides a wizard that takes advantage of this support. The enterprise service discovery wizard uses the resource adapter you specify to query the metadata on an EIS system and use the information it finds to create imports and exports for your service.

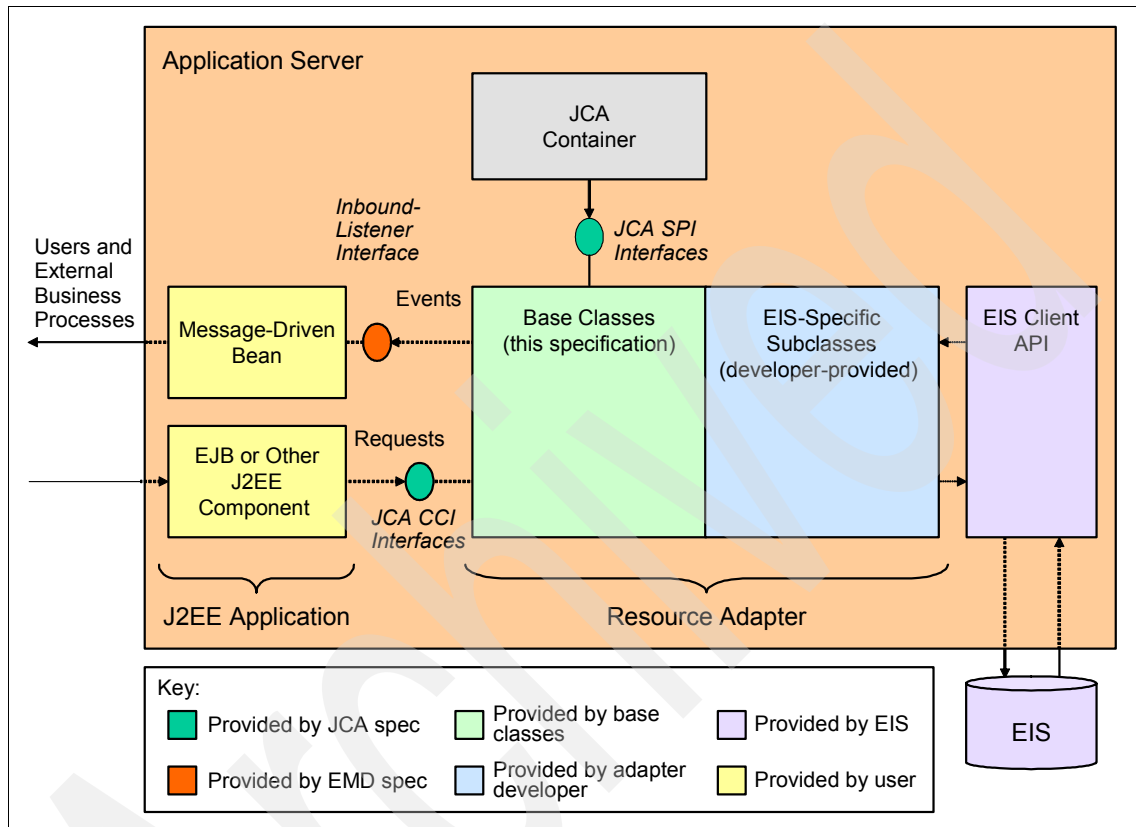


Figure 2-12 WebSphere Adapter architecture

Figure 2-12 illustrates WebSphere Adapter architecture at a high level. In conjunction with the appropriate EIS-specific subclasses, the WebSphere Adapter Foundation Classes provide a JCA-compliant resource adapter implementation that can be managed by the application server to enable connectivity to an EIS.

Outbound requests, those requests intended for the EIS, can be sent to the resource adapter by any J2EE component using the CCI. For inbound requests from the EIS, message-driven beans that implement the InboundListener interface are registered with the adapter by the application-server enabling them to receive any appropriate inbound events from the EIS via the adapter.

2.4 WebSphere Business Integration Adapters

Unlike WebSphere Adapters, WebSphere Business Integration Adapters are not JCA-compliant. As shown in Figure 2-13, WebSphere Business Integration Adapters are distributed. They reside outside of the application server. The server, or integration broker, communicates with this type of adapter through a Java Message Service (JMS) transport layer.

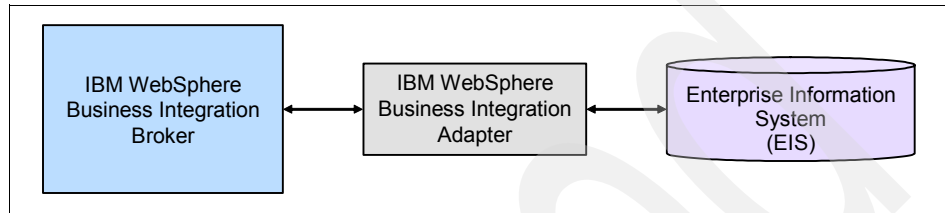


Figure 2-13 IBM WebSphere Business Integration Adapter

Other differences between WebSphere Adapters and WebSphere Business Integration Adapters are:

- ▶ Connection management

WebSphere Adapters rely on standard JCA contracts to manage life cycle tasks such as stopping and starting. WebSphere Business Integration Adapters rely on the WebSphere Adapter Framework to manage connectivity.

- ▶ Event notification

Known as inbound event notification for WebSphere Adapters.

- ▶ Request processing

Known as outbound support in WebSphere Adapters.

- ▶ Object definition

With WebSphere Adapters, you use an Enterprise Metadata Discovery component to probe an EIS and develop business objects and other useful artifacts. This Enterprise Metadata Discovery component is part of the WebSphere Adapter. WebSphere Business Integration Adapters use a separate Object Discovery Agent (ODA) to probe an EIS and generate business object definition schemas.

2.5 IBM WebSphere Adapter versus JCA resource adapter

When developing a custom JCA compliant resource adapter, you can choose to develop either the IBM WebSphere type of resource adapter or the base JCA type of resource adapter.

WebSphere Adapters are fully compliant with JCA 1.5 specification and contain IBM extensions that allows integration with IBM WebSphere Process Server. If you choose to develop an IBM WebSphere type of resource adapter, you can leverage the services provided by the adapter foundation classes. You can extend the generically implemented system contract classes to fit the needs of your custom adapter. Your custom adapter can also make use of the built-in utility APIs to handle common adapter tasks. Using adapter foundation classes significantly reduces your development time and effort to create a custom adapter.

The IBM WebSphere Adapter Toolkit described in 8.2, “WebSphere Adapter Toolkit overview” on page 174 helps you to create both types of resource adapters.

Service Data Objects

Service Data Objects (SDO) are used by WebSphere Process Server to exchange data between service components. It is important to understand SDO in detail. WebSphere Adapter business object model extends SDO. This extension enables data communication between WebSphere Process Server and the adapter. In this chapter we explain:

- ▶ Business objects and SDO
- ▶ WebSphere Adapter business object model
- ▶ WebSphere Adapter business object structure
- ▶ Application-specific information in business objects

3.1 Data: Business Objects and SDO

In any system where heterogeneous components exchange data, there must be an agreed upon data structure or format so that each component knows what to send and what to expect in return.

In Chapter 1, “Introduction and adapter overview” on page 3, we discuss Service Component Architecture (SCA) and touch upon Service Data Object (SDO) slightly. In this section, we discuss SCA in greater detail. We also discuss the concept of business objects. Business objects are the second architectural construct of an SOA (see Figure 3-1). Business data that is exchanged in an integrated application in WebSphere Process Server is represented by constructs called *business objects*. Business objects are based upon a new data access technology called Service Data Objects (SDO).

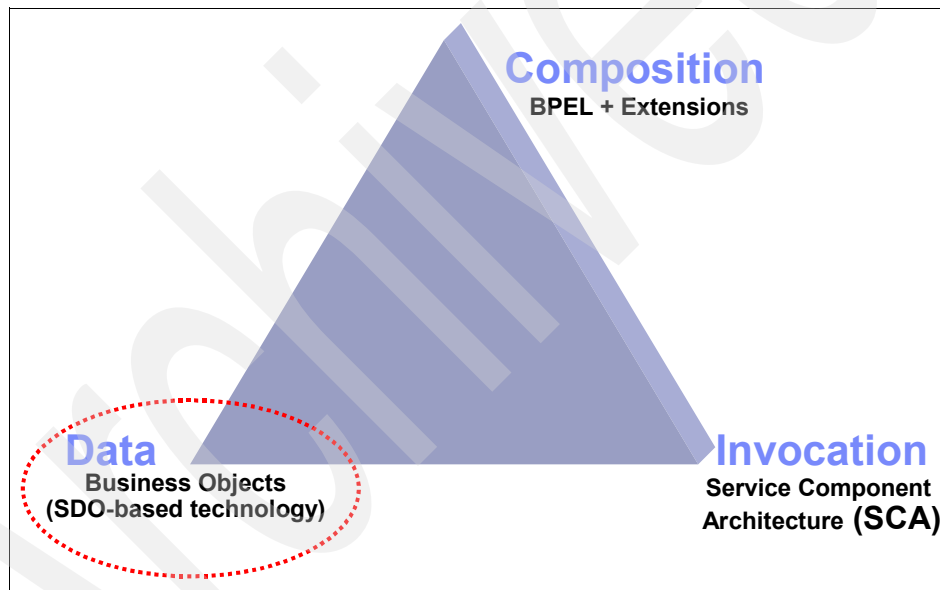


Figure 3-1 Data in SOA: Service Data Objects

3.1.1 SDO design points

When a programmer needs to access data, he or she must select a programming model based on where the data is stored. For example, a Java programmer who wants to manipulate relational data, can use the Java Database Connectivity (JDBC) API. That same API can be used as well to query meta-data. For an Extensible Markup Language (XML) document, the programmer could use for example the Simple API for XML (SAX API).

However, the programmer needs to know and write infrastructure code and that the business logic is tied to the location and type of data. If the data was first stored in an XML data source and is moved to a relational data source, the programmer needs to change the data access code in the program.

SDO changes all that. It unifies data representation across disparate data stores. It supports a disconnected programming model and is integrated with XML. SDO provides dynamic and static (strongly typed) data APIs. The SDO proposal was published jointly by IBM and BEA as Java Specification Request (JSR) 235. SDO V1.0 support is introduced in WebSphere Application Server V6 and IBM Rational Application Developer V6. SDO V2.0 specification is currently available.

Table 3-1 on page 55 lists a number of data-oriented Java APIs and adds the characteristics for SDO as well. For each technology shown in the first column, the table lists some characteristics in the following areas:

- ▶ **Model**
Connected or disconnected are the possible values. This refers to the requirements of some APIs that an active connection to the data source is required. Active connections are incompatible with SOA.
- ▶ **API**
Static or dynamic are the possible values. Static code is generated in advance while dynamic code has a reflective interface that avoids code generation. However, dynamic code lacks compile-time checking.
- ▶ **Data Source**
This refers to the intrinsic nature of the data. SDO works uniformly across all kinds of data sources.
- ▶ **Meta-data API**
This refers to the API that can be used to obtain type information about the data itself.
- ▶ **Query Language**
This column specifies what option is available to define the data that is accessed by a service. SDO has the ability to handle any query language.

Table 3-1 Comparing data-oriented Java APIs

Technology	Model	API	Data Source	Meta-data API	Query Language
SDO	Disconnected	Dynamic and static	Any	SDO Meta-data API and Java Introspection	Any

Technology	Model	API	Data Source	Meta-data API	Query Language
JDBC Rowset	Connected	Dynamic	Relational	Relational	SQL
JDBC Cached Rowset	Disconnected	Dynamic	Relational	Relational	SQL
Entity EJB™	Connected	Static	Relational	Java Introspection	EJBQL
JDO	Connected	Static	Relational, Object	Java Introspection	JDOQL
JCA	Disconnected	Dynamic	Record-based	Undefined	Undefined
DOM and SAX	Disconnected	Dynamic	XML	XML InfoSet	XPath, XQuery
JAXB	Disconnected	Static	XML	Java Introspection	N/A
JAX-RPC	Disconnected	Static	XML	Java Introspection	N/A

In addition to providing a programming model that unifies data access, there are several other key design features to note about SDO. First, built into the SDO architecture is support for some common programming patterns. Most significantly, SDO supports a disconnected programming model. Typically with this type of pattern a client can be disconnected from a particular data access service (DAS) while working with a set of business data.

However, when the client has completed processing and needs to apply changes to a back-end data store by way of a DAS, a change summary is necessary in order to provide the appropriate level of data concurrency control. This change summary information has been built into the SDO programming model to support this common data access scenario. Another important design point to note is that SDO integrates well with XML. As a result, SDO naturally fits in with distributed service-oriented applications where XML plays a very important role.

Finally, SDO has been designed to support both dynamic and static data access APIs. The dynamic APIs are provided with the SDO object model and provide an interface that allows developers to access data even when the schema of the data is not known until runtime. In contrast to this, the static data APIs are used when the data schema is known at development time, and the developer prefers to work with strongly typed data access APIs.

3.1.2 Some SDO concepts

Prior to moving into the details of the business object architecture, it is useful to understand a few key SDO concepts that are important to understand before discussing the design and use of business objects in WebSphere Process Server.

The fundamental concept in the SDO architecture is the data object. In fact, it is not uncommon to see the term SDO used interchangeably with the term data object. A *data object* is a data structure that holds primitive data, multi-valued fields (other data objects), or both. The data object also holds references to metadata that provides information about the data included in the data object. In the SDO programming model, data objects are represented by the `commonj.sdo.DataObject` Java interface definition.

This interface includes method definitions that allow clients to get and set the properties associated with the `DataObject`. As an example, consider modeling customer data with an SDO data object. The properties associated with the customer might be things such as, `firstName` (`String`), `lastName` (`String`), and `customerID` (`long`).

Example 3-1 displays pseudocode that shows how you would use the `DataObject` API to get and set properties for the customer data object.

Example 3-1 pseudo code to set properties for customer data object

```
DataObject customer = ...  
customer.setString("firstName", "John");  
customer.setString("lastName", "Doe");  
customer.setInt("customerID", 123);  
int id = customer.getInt("customerID");
```

Another important concept in the SDO architecture is the data graph. A *data graph* is a structure that encapsulates a set of data objects. From the top-level data object contained in the graph, all other data objects are reachable by traversing the references from the root data object. Another important feature included in the data graph is a change summary that is used to log information about what data objects in the graph have changed during processing.

In the SDO programming model, data graphs are represented by the `commonj.sdo.DataGraph` Java interface definition. In addition to this, the change summary information is defined by the `commonj.sdo.ChangeSummary` interface as Figure 3-2 shows.

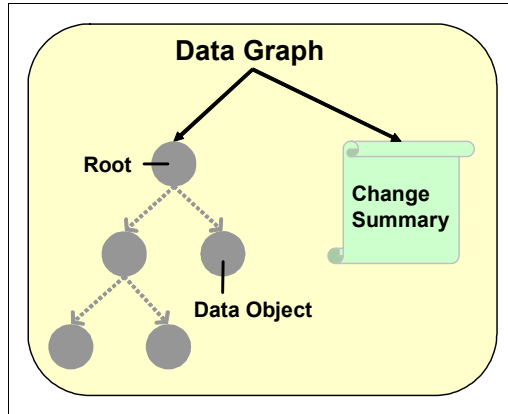


Figure 3-2 Data graph consisting of data objects and change summary

3.1.3 Business objects and the business object framework

The business object framework is intended to provide a data abstraction for the Service Component Architecture (SCA). For readers that have experience with the InterChange Server, the business object framework in WebSphere Process Server provides capabilities similar to business objects as used in the InterChange Server. Business objects are based on SDO v1.0 technology but provide some additional functionality not found in SDO.

Both SCA and SDO (the basis of business objects) have been designed to be complimentary service oriented technologies. Figure 3-3 on page 59 illustrates how SCA provides the framework to define service components and compose these services into integrated application, and it further shows that business objects represent the data that flows between each service. Whether the interface associated with a particular service component is defined as a Java interface or a WSDL port type, the input and output parameters are represented using business objects.

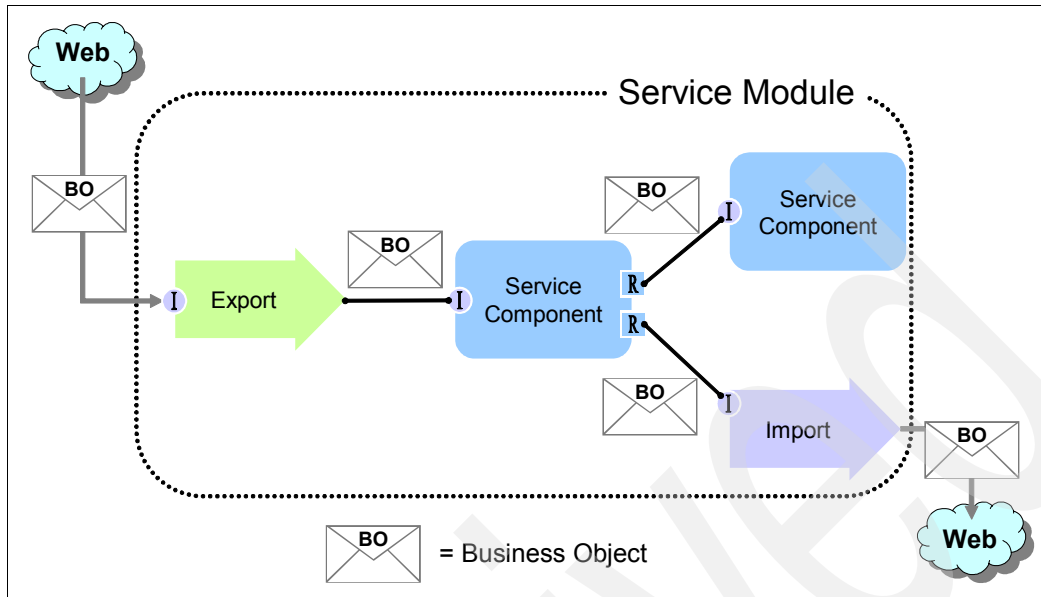


Figure 3-3 Exchanging data in an SCA runtime

The business object framework consists of the following four concepts:

- ▶ Business object (BO), a fundamental data structure for representing business data
- ▶ Business graph (BG), wrapper for a business object or hierarchy of business objects to provide enhanced information such as change summary, event summary and verb
- ▶ BO Type Metadata, providing the ability to annotate business objects with application-specific information.
- ▶ BO Services provided to facilitate working with business objects

These services are available in addition to the capabilities already provided by SDO V1.0.

The *business object* is the fundamental data structure that represents business data.

Note: It is not uncommon to hear the term business object used to refer to the entire framework. However, in this book when we use the term *business object*, we refer to the fundamental data *structure* for representing business data and not to the overall architecture.

When discussing the overall architecture, we use the term *business object framework*. The business object is directly related to the SDO data object concept discussed previously. In fact, business objects in WebSphere Process Server are represented in memory with the SDO type `commonj.sdo.DataObject`. Business objects are modeled as XML schemas.

Two types of business objects can be considered:

- ▶ Simple business objects are composed only of scalar properties.
- ▶ Hierarchical business objects are composed of attributes that refer to nested business objects.

The business graph is used in the business object framework to wrap a top-level business object and provide additional information to enhance the data. In particular, the data graph includes the change summary for the data in the graph (similar to the SDO change summary information), event summary, and verb information (used for data synchronization between Enterprise Information Systems). The business graph is very similar to the concept of the data graph in the SDO architecture. However, the event summary and the verb portion of the enhanced information are not included with the SDO data graph.

Figure 3-4 on page 61 shows graphically how all these concepts fit together. At the bottom, we have the simple business object `Stock`. This business object has only scalar attributes. The business object `Customer` is a hierarchical business object. It has scalar attributes, but it also has an attribute that is an array of business objects (`BO Stock` in this case). Business graph `CustomerBG` acts as a wrapper containing the top-level business object. Because `CustomerBG` is a business graph, it has the additional attribute `Verb` and a reference to a change summary and event summary. The change summary concept is inherited from SDO, while the concept of a verb and event summary are provided by WebSphere Process Server itself.

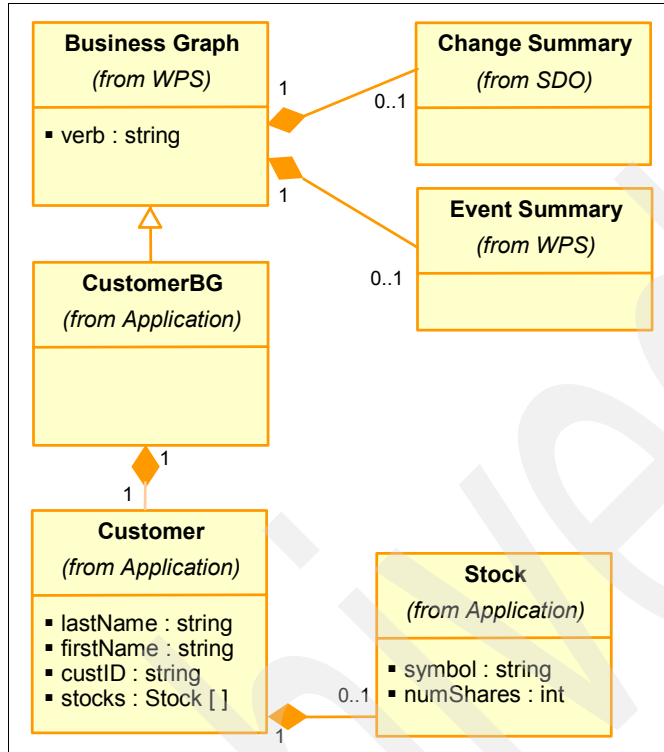


Figure 3-4 Business graph and business objects

3.2 WebSphere Adapter business object model

WebSphere adapters and the custom adapter developed in this book use the WebSphere Adapter business object model for data communication. At the technical level, the WebSphere Adapter business object model maps directly to the SDO model; a business graph and business object in WebSphere adapter business object world map to a data graph and data object in the SDO domain.

Note: The WebSphere Adapter business object provides extensions and additional functionality on top of the SDO model that currently does not allow adapters to consume data based on the SDO 1.0 model alone.

A business graph is considered a top-level structure and should define a single child business object which can itself contain zero or more child business objects. The business graph acts to hold metadata about its child business

objects. See example of WebSphere Adapter business object model in Figure 3-5.

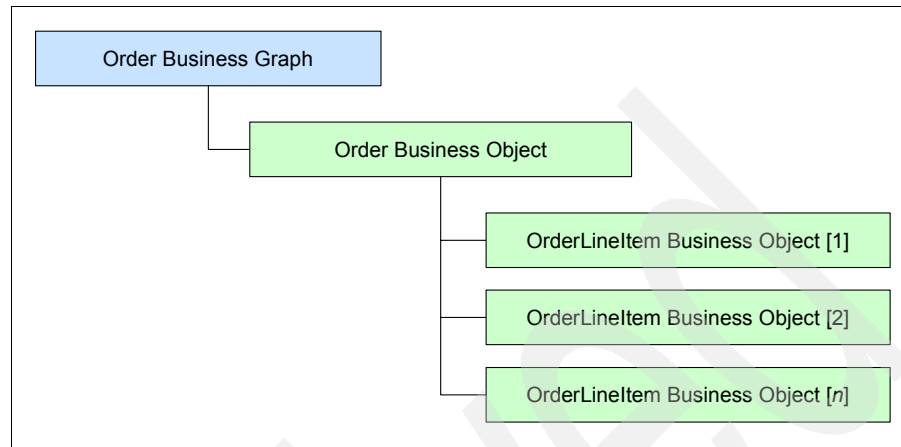


Figure 3-5 WebSphere Adapter Business Object model

Note: The term business object is often used to reflect a higher-level notion of this larger combined structure (business graph in addition to any contained business objects). Unless it is used in the context of a business graph as in this section, it is best to interpret *business object* to mean the entire structure.

In JCA 1.5 specification an optional CCI Record model is defined for data communication, the WebSphere Adapter business object model has several advantages over CCI Record model:

- ▶ The WebSphere Adapter business object model provides a full, working implementation as opposed to the CCI Record model which simply defines interfaces that must be implemented by the adapter developer.
- ▶ The WebSphere Adapter business object model provides built-in support for tracking changes at both the object- and property-level, allowing for improved processing efficiency and reduced bandwidth requirements for exchanging data.
- ▶ The WebSphere Adapter business object model is based upon the open-standard Service Data Object (SDO) model, which is supported by IBM and others. For more information, see:

<http://www.eclipse.org/>

- ▶ The WebSphere Adapter business object model aligns well with the larger WebSphere SOA strategy which, going forward, better enables interoperability with other WebSphere-based applications.

3.2.1 After-images versus deltas

WebSphere Adapter business object model has two distinct types of business objects:

- ▶ After-images

After-image business objects can be thought of as a snapshot of the data in time. They reflect how the EIS entity looks (for inbound events) or is expected to look (for outbound requests). An after-image business object should represent the entire entity structure in the EIS. For example, if an Order object with Update verb is sent to the adapter, and the order has two ORDER_LINE children, the object in the EIS should be modified to reflect the input. If the EISs representation of that object had only one ORDER_LINE, it then has two after the processing has completed.

- ▶ Deltas

Delta business objects reflect the specific changes that have occurred (for inbound events) or that the user wants affected (for outbound requests) in the EIS. Each business object contains a change Summary structure that can store all the pending changes. For outbound requests, the adapter must interpret the change summary, making all applicable changes to the data. For example, if an ORDER_LINE has been added to an Order object, the ORDER_LINE appears as Created in the change summary. The adapter is responsible for finding that Order, and adding the ORDER_LINE to it.

The two types are used to convey different information.

3.2.2 Verbs

In WebSphere Adapter business object model, all business object graph definitions (whether or not they are ever used to hold after-image data) should define a property to hold a verb. Whereas with a delta business object, a component can introspect the change summary to determine what actually occurred, with an after-image, there is no inherent information to let a component determine how to interpret the object.

This creates a problem for components that require such information to act. For example, a component that encountered an after-image business object of an Order, would require additional information to differentiate whether the business object represented a newly created entity in the EIS or simply an existing entity that was updated. For this reason, after-image business objects require the

inclusion of a verb property. A *verb property* is one defined in the business graph definition that allows downstream components to interpret what action was or is intended to be performed on the business object.

3.2.3 Verbs versus operations

Verbs and operations are similar concepts but serve different purposes. *Operations* reflect the functions that an adapter can perform. An operation is directly related to the adapter performing it, while a *verb* is directly related to the business object in which it is specified. In general, the verb defined for a business object should match the operation, but not always. Some operations are unrelated to the verb. For example, if you wanted to create a new entity in the EIS using an after-image business object, the object would be expected to have a verb of Create and the adapter would need to be invoked with the **Create** operation. You would not invoke the adapter's Delete operation with an object that had a Create verb. This would be illogical and, in fact, would result in an error reported by the adapter.

However, some operations, specifically those that do not fall under Create, Retrieve, Update, or Delete (CRUD), do not require verbs. For example, the Retrieve operation of adapters, which has the adapter query the EIS to find an entity that matches the business object, does not expect a verb. This Retrieve operation is unconcerned with what action lead to the business object being created (for example, the information reflected by the verb), it is merely concerned with the retrieving entity in the EIS reflect by the object.

3.2.4 Business object naming

Business objects should be named to reflect the structure they represent: for example, Customer or Address. Names are most likely be derived during the metadata import process based on the name given by the EIS.

Business object names should be converted to *camel case* (that is, remove separators such as spaces or underscores and capitalize the first letter of each word) from EIS name. For example, ORDER_LINE_ITEM becomes OrderLineItem. According to the WebSphere Adapter business object specification, the parent business object graph should be named for the contained business object followed by BG, CustomerBG for a Customer business object, for example. Business object names as well as property names should have no semantic value to the adapter. When you develop your adapter logic, be sure the logic is based on metadata, as opposed to naming conventions.

3.2.5 Blank, Ignore, and Null properties in business objects

WebSphere Adapter business object model support the notion of blank, ignore, and null properties.

Blank property

For blank properties, you must reset the value in the EIS to the default. According to the WebSphere Adapter BO model, *blank properties* are identified as those in which the BO property is not set and no information is included for the property in the ChangeSummary. Request processing should be done as follows: if BO defines a default value for property, set the corresponding property in EIS entity to this default value. Otherwise, set the property in the EIS entity to Java default value for property type (for example, zero-length string for strings, 0 for numbers and so on).

Ignore property

For ignore properties, you must set to ignore or leave the existing value as is. As for the WebSphere Adapter BO model, ignore properties are identified as those in which the BO property is not set, but ChangeSummary is updated for this property. Request processing should be done as follows: For all operations, adapters should simply not modify the given property.

Null property

If ChangeSummary is included and the EIS supports the notion of a null value, a null should be set in the corresponding EIS entity. However, the EIS does not support null values, blank semantics should be followed to clear or reset the corresponding property in the EIS entity. If ChangeSummary is not included, the adapter is unable to distinguish between explicitly set null, ignore or blank values. The default action established for adapters in this case is to follow the ignore semantics. This means that ChangeSummary must be included on a BO request if you want to reflect explicit null and blank values.

3.3 Business object structure

For applications that use databases, each business object can correspond to a database table or view, and each simple attribute within the object corresponds to a column in that table or view.

Business objects can be flat or hierarchical. All of the attributes of a *flat business object* are simple and represent one row in the database table. The term *hierarchical business object* refers to a complete business object, including all the child business objects that it contains at any level. The term *individual business object* refers to one business object, independent of child business

objects that it might contain or that contain it. The individual business object can represent a view that spans multiple database tables. The term *top-level business object* refers to the individual business object at the top of the hierarchy that does not itself have a parent business object.

A hierarchical business object has attributes that represent a child business object, an array of child business objects, or a combination of the two. In turn, each child business object can contain a child business object or an array of business objects, and so on.

A single-cardinality relationship occurs when an attribute in a parent business object represents one child business object. In this case, the attribute is of the same type as the child business object.

A multiple-cardinality relationship occurs when an attribute in the parent business object represents an array of child business objects. In this case, the attribute is of the same type as the child business objects. The adapter supports the following relationships among business objects:

- ▶ Single-cardinality relationships
- ▶ Single-cardinality relationships and data without ownership
- ▶ Multiple-cardinality relationships in each type of cardinality

The relationship between the parent and child business objects is described by the application-specific information of the key attribute in the business object storing the relationship.

3.3.1 Single-cardinality relationships

Typically, a business object that contains a single-cardinality child business object has at least two attributes that represent the relationship. The type of one attribute is the same as the child's type. The other attribute is a simple attribute that contains the child's primary key as a foreign key in the parent. The parent has as many foreign-key attributes as the child has primary-key attributes. Because the foreign keys that establish the relationship are stored in the parent, each parent can contain only one child business object of a given type. See Figure 3-6 on page 67. In the example, FKey in the ParentBOName box is the simple attribute that contains the child's primary key, and Child(1), also in the ParentBOName box, is the attribute that represents the child business object.

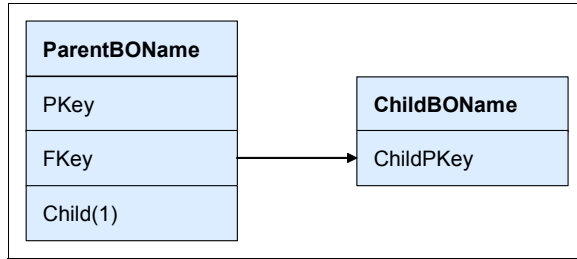


Figure 3-6 Typical single cardinality relationship

3.3.2 Single-cardinality relationships and data without ownership

Typically, each parent business object owns the data within the child business object that it contains. For example, if each Customer business object contains one Address business object, when a new customer is created, a new row is inserted into both the customer and address tables. The new address is unique to the new customer.

Likewise, when deleting a customer from the customer table, the customer's address is also deleted from the address table. However, situations can occur in which multiple hierarchical business objects contain the same data, which none of them owns. For example, assume that an Address business object has a StateProvince attribute that represents the StateProvince lookup table with single cardinality. Because the lookup table is rarely updated and is maintained independently of the address data, creation or modification of address data does not affect the data in the lookup table.

The adapter either finds an existing state or province name or fails. It does not add or change values in the lookup table. When multiple business objects contain the same single-cardinality child business object, the foreign-key attribute in each parent business object must specify the relationship as NOOWNERSHIP.

3.3.3 Multiple-cardinality relationships in business objects

Typically, a business object that contains an array of child business objects has only one attribute representing the relationship. The type of the attribute is an array of the same type as the child business objects. For a parent to contain more than one child, the foreign keys that establish the relationship are stored in the child. Therefore, each child has at least one simple attribute that contains the parent's primary key as a foreign key. The child has as many foreign-key

attributes as the parent has primary key attributes. Because the foreign keys that establish the relationship are stored in the child, each parent can have zero or more children.

Figure 3-7 illustrates a multiple-cardinality relationship. In the example, ParentID in the three ChildBOName boxes is the simple attribute that contains the parent's primary key. Child1 in the ParentBOName box is the attribute representing the array of child business objects.

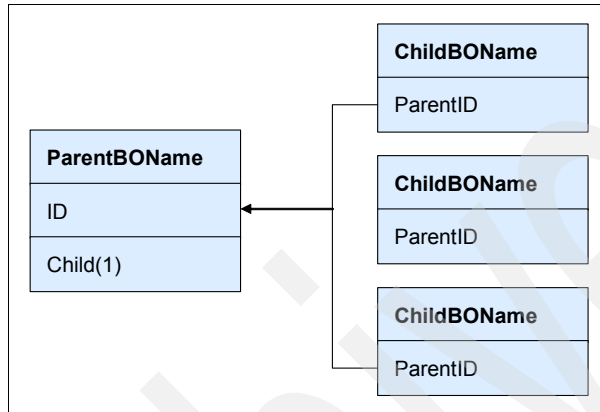


Figure 3-7 Typical multiple cardinality relationship

A multiple cardinality relationship could be an N=1 relationship. Some applications store one child entity so that the parent-child relationship is stored in the child rather than in the parent. In other words, the child contains a foreign key whose value is identical to the value stored in the parent's primary key. Applications use this type of relationship when child data does not exist independently of its parent and can be accessed only through its parent. Such child data requires that the parent and its primary key value exist before the child and its foreign-key value can be created.

Figure 3-8 shows this Multiple cardinality relationship with N=1 relationship.

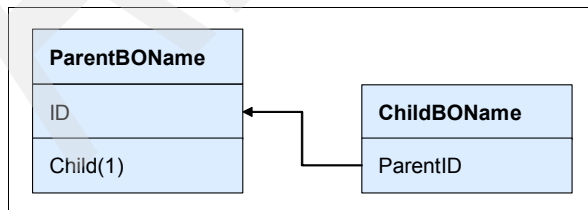


Figure 3-8 Multiple cardinality relationship with N=1.

3.4 Application-specific information (ASI)

Application-specific information in business object definitions provides the adapter with application-dependent instructions on how to process business objects. The adapter parses the application-specific information from the attributes or verb of a business object or from the business object itself to generate queries for Create, Retrieve, Update and Delete operations. The ASI definitions are typically save an xsd schema file. This schema file is used to constrain the ASI elements in the business object definition.

3.4.1 Application-specific information at the business-object level

Application-specific information at the business-object level is normally used to specify the name of the corresponding database table and to provide information necessary to perform a physical or logical delete operation. Example 3-2 shows a sample business object level ASI definition in the schema file jdbcasi.xsd for JDBC adapter.

Example 3-2 Sample JDBC adapter business object level ASI definition

```
<jdbcasi:TableName>customer</jdbcasi:TableName>
<jdbcasi:StatusColumnName>status</jdbcasi:StatusColumnName>
<jdbcasi::StatusValue>deleted</jdbcasi:StatusValue>
```

3.4.2 Application-specific information at the Verb level

Stored procedures are defined at the verb level. Each stored procedure definition consists of the follow elements: StoredProcedureType, StoredProcedureName, ResultSet, and Parameters. See Example 3-3 for a sample of stored procedure definition for JDBC adapter.

Example 3-3 Sample JDBC ASI schema with stored procedure defined at verb level

```
<jdbcasi:JDBCBusinessObjectTypeMetadata
xmlns:jdbcasi="http://www.ibm.com/xmlns/prod/websphere/j2ca/jdbc/metadata">
  <jdbcasi:TableName>customer</jdbcasi:TableName>
  <jdbcasi:Operation>
    <jdbcasi:Name>Retrieve</jdbcasi:Name>
    <jdbcasi:StoredProcedures>
      <jdbcasi:StoredProcedureType>RetrieveSP</jdbcasi:StoredProcedureType>
      <jdbcasi:StoredProcedureName>retrieve_cust</jdbcasi:StoredProcedureName>
      <jdbcasi:ResultSet>>false</jdbcasi:ResultSet>
      <jdbcasi:Parameters>
        <jdbcasi:Type>IP</jdbcasi:Type>
        <jdbcasi:PropertyName>primaryKey</jdbcasi:PropertyName>
      </jdbcasi:Parameters>
    </jdbcasi:StoredProcedures>
  </jdbcasi:Operation>
</jdbcasi:JDBCBusinessObjectTypeMetadata>
```

```

<jdbcasi:Parameters>
  <jdbcasi:Type>OP</jdbcasi:Type>
  <jdbcasi:PropertyName>custCode</jdbcasi:PropertyName>
</jdbcasi:Parameters>
<jdbcasi:Parameters>
  <jdbcasi:Type>OP</jdbcasi:Type>
  <jdbcasi:PropertyName>firstName</jdbcasi:PropertyName>
</jdbcasi:Parameters>
<jdbcasi:Parameters>
  <jdbcasi:Type>OP</jdbcasi:Type>
  <jdbcasi:PropertyName>lastName</jdbcasi:PropertyName>
</jdbcasi:Parameters>
</jdbcasi:StoredProcedures>
<jdbcasi:StoredProcedures>
  <jdbcasi:StoredProcedureType>AfterRetrieveSP</jdbcasi:StoredProcedureType>
  <jdbcasi:StoredProcedureName>retrieve_cust</jdbcasi:StoredProcedureName>
  <jdbcasi:ResultSet>false</jdbcasi:ResultSet>
  <jdbcasi:Parameters>
    <jdbcasi:Type>IP</jdbcasi:Type>
    <jdbcasi:PropertyName>primaryKey</jdbcasi:PropertyName>
  </jdbcasi:Parameters>
  <jdbcasi:Parameters>
    <jdbcasi:Type>OP</jdbcasi:Type>
    <jdbcasi:PropertyName>custCode</jdbcasi:PropertyName>
  </jdbcasi:Parameters>
  <jdbcasi:Parameters>
    <jdbcasi:Type>OP</jdbcasi:Type>
    <jdbcasi:PropertyName>firstName</jdbcasi:PropertyName>
  </jdbcasi:Parameters>
  <jdbcasi:Parameters>
    <jdbcasi:Type>OP</jdbcasi:Type>
    <jdbcasi:PropertyName>lastName</jdbcasi:PropertyName>
  </jdbcasi:Parameters>
</jdbcasi:StoredProcedures>
</jdbcasi:Operation>
</jdbcasi:JDBCBusinessObjectTypeMetadata>

```

3.4.3 Application-specific information at the attribute level

Application-specific information at the attribute level is typically used to specify the meta-data for an attribute. For example, we can use ASI to specify that an attribute is a primary key. See Example 3-4.

Example 3-4 Sample attribute level ASI for JDBC adapter

```

<jdbcasi:ColumnName>pkey</jdbcasi:ColumnName>
<jdbcasi:PrimaryKey>true</jdbcasi:PrimaryKey>

```



```

<jdbcasi:FixedChar>true</jdbcasi:FixedChar>
</jdbcasi:JDBCAttributeTypeMetadata>
</appinfo>
</annotation>
<simpleType>
  <restriction base="string">
    <maxLength value="10"/>
  </restriction>
</simpleType>
</element>
<element name="custCode" type="string">
  <annotation>
    <appinfo source="WBI">
      <jdbcasi:JDBCAttributeTypeMetadata
xmlns:jdbcasi="http://www.ibm.com/xmlns/prod/websphere/j2ca/jdbc/metadata">
        <jdbcasi:ColumnName>ccode</jdbcasi:ColumnName>
        <jdbcasi:ForeignKey>custinfoObj/custCode</jdbcasi:ForeignKey>
      </jdbcasi:JDBCAttributeTypeMetadata>
    </appinfo>
  </annotation>
</element>
<element name="firstName" type="string">
  <annotation>
    <appinfo source="WBI">
      <jdbcasi:JDBCAttributeTypeMetadata
xmlns:jdbcasi="http://www.ibm.com/xmlns/prod/websphere/j2ca/jdbc/metadata">
        <jdbcasi:ColumnName>fname</jdbcasi:ColumnName>
      </jdbcasi:JDBCAttributeTypeMetadata>
    </appinfo>
  </annotation>
</element>
<element name="lastName" type="string">
  <annotation>
    <appinfo source="WBI">
      <jdbcasi:JDBCAttributeTypeMetadata
xmlns:jdbcasi="http://www.ibm.com/xmlns/prod/websphere/j2ca/jdbc/metadata">
        <jdbcasi:ColumnName>lname</jdbcasi:ColumnName>
      </jdbcasi:JDBCAttributeTypeMetadata>
    </appinfo>
  </annotation>
</element>

```

Outbound request processing

This chapter describes the conceptual view of an outbound request processing of an adapter. It also includes the following topics:

- ▶ Outbound request processing
- ▶ Adapter clients
- ▶ Service input and output data
- ▶ Adapter service interface
- ▶ EIS service import
- ▶ Adapter configuration properties
- ▶ Application sign-on
- ▶ Connection management
- ▶ Outbound interaction
- ▶ Standard outbound operations and processing
- ▶ Command Pattern
- ▶ Transaction support

4.1 Outbound request processing

Let us consider the scenario where an application component wishes to invoke an operation on the Enterprise Information System (EIS). For example, the application component wishes to retrieve data from the EIS. One way to do this is to write custom code in the application component to connect and invoke the EIS retrieve operation. As explained in Chapter 2, “Adapter basics” on page 33, this creates point-to-point integration and is not desirable. A better method is to develop a JCA-compliant adapter. The adapter acts as the connector between the application component and the EIS. By using a J2EE Connector Architecture (JCA) compliant adapter, the application component can rely on the adapter to do the following:

- ▶ The adapter, in collaboration with application server, handles connection and connection pooling to the EIS.
- ▶ The adapter provides a standard interface for the application component. The application component does not need to know the EIS-specific interface/APIs.
- ▶ The adapter accepts standard data objects from the application component. The application component does not need to know or care about EIS-specific objects.
- ▶ The adapter returns standard data objects that encapsulate the result of the EIS operation invocation to the application component. The conversion of EIS-specific objects to standard data objects is handled by the adapter.

We can define adapter outbound request processing as a process that involves the following:

- ▶ Implement the JCA connection management contract to handle connections to the EIS and delegate connection pooling to the application server.
- ▶ Specify connection and security properties in the adapter deployment descriptor.
- ▶ Provide a standard interface that accepts business objects.
- ▶ Provide operations in the service interface for the application components to invoke.
- ▶ Convert business objects from application components to EIS-specific objects used in EIS operation invocations.
- ▶ Map application-invoked adapter operations to EIS specific operations.
- ▶ Invoke EIS operations on behalf of application components.
- ▶ Convert EIS objects returned from EIS operations to business objects.
- ▶ Return the result of the adapter operation to the application component using business objects.

- Provide local or XA transaction support for adapter outbound operations.
- Log any errors or exceptions that may occur.

The details of outbound request processing and what needs to be done to process an outbound request are discussed next. Because the adapter we develop in this book is intended to run on WebSphere Process Server, this process is described in the context of Service Component Architecture (SCA).

4.2 Adapter clients

Adapter clients are application components running on the application server. In WebSphere Process Server an adapter client can be an SCA service component wired to the adapter through an EIS service import. JSPs or servlets can also invoke adapter outbound operations when wired through an SCA standalone reference to an EIS service import. Figure 4-1 shows an SCA service component connected to the adapter through an EIS service import for outbound operations to the EIS.

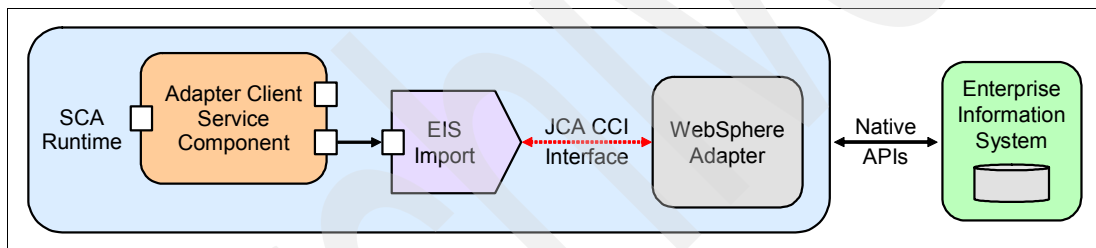


Figure 4-1 Connecting to EIS through EIS import and adapter

The SCA client programming model provides facilities to locate a service, to create data objects, to invoke an exception-handling service for those exceptions raised by the invoked component.

Clients locate services by using the `ServiceManager` class. There are a few ways to instantiate the `ServiceManager` class, depending on the desired lookup scope for the service. After the `ServiceManager` is instantiated, you can use the `locateService` (String interface) method to locate the service that implements the requested interface.

The SCA supports both static (type-safe) and dynamic invocation. The snippet of pseudo-code in Example 4-1 on page 76 shows how the service is located and how the `someOperation` method is then invoked dynamically. The dynamic invocation interface provides an `invoke` method for for this. The `invoke` method

takes the name of the method to invoke and an array of objects as input. The invoke method returns an array of objects. Ideally, the input is a Service Data Objects (SDO), however ordinary Java classes can also be used.

Example 4-1 Dynamic service invocation

```
Service myService = (Service) serviceManager.locateService("myService");
DataObject input = ...
myService.invoke("someOperation", input);
```

The pseudo-code snippet in Example 4-2 shows the type-safe invocation.

Example 4-2 Static (type-safe) invocation

```
MyServiceImpl myService =
    (MyServiceImpl) serviceManager.locateService("myService");
myService.someMethod("input");
```

4.3 Service input and output data

Outbound data is sent from the application component to the adapter as a business graph. In the business object framework, a business graph is a wrapper for a business object or hierarchy of business objects that provides enhanced information such as change summary, event summary, and verb. For an outbound request-response adapter operation, a business graph is returned to the calling application component. This business graph contains the result of outbound operation on the EIS. For example, if the outbound operation is a Retrieve to retrieve data from the EIS, then the return business graph is populated with retrieved data.

For the application component to interact with the EIS through the adapter, you must create the business object and business graph definitions. They can be created manually, or if the adapter implements Enterprise Metadata Discovery (EMD) they can be mined automatically from the underlying EIS using the Enterprise Service Discovery wizard in WebSphere Integration Developer.

Note: The JCA specification specifies a `javax.resource.cci.Record` object for application components to exchange data with the adapter. WebSphere Foundation Classes provides a `WBIRRecord` object to wrap the business object. A business graph from the application component is bound to the `WBIRRecord` object through the EIS service import.

4.4 Adapter service interface

An adapter service interface is defined by a WSDL port type. Arguments and return values are described using the business object definition XML schema. An adapter exposes its available outbound operations to the client through its service interface. The adapter interface defines the operations that can be called and the data that is passed, such as input arguments, and returned values.

An adapter service interface can be created using WebSphere Integration Developer's interface editor or if the adapter implements Enterprise Metadata Discovery (EMD) it can be created automatically using the Enterprise Service Discovery wizard.

An example of the interface WSDL file is shown in Example 4-3.

Example 4-3 Example of a service interface file

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:bons1="http://www.ibm.com/xmlns/prod/wbi/j2ca/redmaintenance/apartment"
  xmlns:tns="http://RedMaintenanceModule/interfaces/RedMaintenanceOutboundInterface"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  name="RedMaintenanceOutboundInterface"
  targetNamespace="http://RedMaintenanceModule/interfaces/RedMaintenanceOutboundInterface">
  <wsdl:types>
    <xsd:schema
      targetNamespace="http://RedMaintenanceModule/interfaces/RedMaintenanceOutboundInterface"
      xmlns:bons1="http://www.ibm.com/xmlns/prod/wbi/j2ca/redmaintenance/apartment"
      xmlns:tns="http://RedMaintenanceModule/interfaces/RedMaintenanceOutboundInterface"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import
        namespace="http://www.ibm.com/xmlns/prod/wbi/j2ca/redmaintenance/apartment"
        schemaLocation="../../xsd-includes/http.www.ibm.com.xmlns.prod.wbi.j2ca.redmaintenance.apartment.xsd"/>
      <xsd:element name="createApartment">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="createApartmentInput" nillable="true"
              type="bons1:ApartmentBG"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="createApartmentResponse">
        <xsd:complexType>
          <xsd:sequence>
```

```

        <xsd:element name="createApartmentOutput" nillable="true"
type="bons1:ApartmentBG"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
    <xsd:element name="deleteApartment">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="deleteApartmentInput" nillable="true"
type="bons1:ApartmentBG"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
        <xsd:element name="deleteApartmentResponse">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="deleteApartmentOutput" nillable="true"
type="bons1:ApartmentBG"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
            <xsd:element name="updateApartment">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="updateApartmentInput" nillable="true"
type="bons1:ApartmentBG"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
                <xsd:element name="updateApartmentResponse">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="updateApartmentOutput" nillable="true"
type="bons1:ApartmentBG"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                    <xsd:element name="retrieveApartment">
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:element name="retrieveApartmentInput" nillable="true"
type="bons1:ApartmentBG"/>
                            </xsd:sequence>
                        </xsd:complexType>
                    </xsd:element>
                        <xsd:element name="retrieveApartmentResponse">
                            <xsd:complexType>
                                <xsd:sequence>

```



```

        <xsd:element name="retrieveApartmentOutput" nillable="true"
type="bons1:ApartmentBG"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
</wsdl:types>
    <wsdl:message name="createApartmentRequestMsg">
        <wsdl:part element="tns:createApartment" name="createApartmentParameters"/>
    </wsdl:message>
    <wsdl:message name="createApartmentResponseMsg">
        <wsdl:part element="tns:createApartmentResponse"
name="createApartmentResult"/>
    </wsdl:message>
    <wsdl:message name="deleteApartmentRequestMsg">
        <wsdl:part element="tns:deleteApartment" name="deleteApartmentParameters"/>
    </wsdl:message>
    <wsdl:message name="deleteApartmentResponseMsg">
        <wsdl:part element="tns:deleteApartmentResponse"
name="deleteApartmentResult"/>
    </wsdl:message>
    <wsdl:message name="updateApartmentRequestMsg">
        <wsdl:part element="tns:updateApartment" name="updateApartmentParameters"/>
    </wsdl:message>
    <wsdl:message name="updateApartmentResponseMsg">
        <wsdl:part element="tns:updateApartmentResponse"
name="updateApartmentResult"/>
    </wsdl:message>
    <wsdl:message name="retrieveApartmentRequestMsg">
        <wsdl:part element="tns:retrieveApartment"
name="retrieveApartmentParameters"/>
    </wsdl:message>
    <wsdl:message name="retrieveApartmentResponseMsg">
        <wsdl:part element="tns:retrieveApartmentResponse"
name="retrieveApartmentResult"/>
    </wsdl:message>
    <wsdl:portType name="RedMaintenanceOutboundInterface">
        <wsdl:operation name="createApartment">
            <wsdl:input message="tns:createApartmentRequestMsg"
name="createApartmentRequest"/>
            <wsdl:output message="tns:createApartmentResponseMsg"
name="createApartmentResponse"/>
        </wsdl:operation>
        <wsdl:operation name="deleteApartment">
            <wsdl:input message="tns:deleteApartmentRequestMsg"
name="deleteApartmentRequest"/>
            <wsdl:output message="tns:deleteApartmentResponseMsg"
name="deleteApartmentResponse"/>
        </wsdl:operation>

```

```

        <wsdl:operation name="updateApartment">
            <wsdl:input message="tns:updateApartmentRequestMsg"
name="updateApartmentRequest"/>
            <wsdl:output message="tns:updateApartmentResponseMsg"
name="updateApartmentResponse"/>
        </wsdl:operation>
        <wsdl:operation name="retrieveApartment">
            <wsdl:input message="tns:retrieveApartmentRequestMsg"
name="retrieveApartmentRequest"/>
            <wsdl:output message="tns:retrieveApartmentResponseMsg"
name="retrieveApartmentResponse"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>

```

4.5 EIS service import

The EIS service import is an SCA import that allows adapter client components in the SCA module to use an EIS service defined outside of the SCA module. The EIS import binding binds the external EIS service to the SCA module. It specifies the mapping between the definition of the outbound operation invocation and the interaction information as understood by the adapter using JCA interfaces. The EIS import binding happens at three levels:

- ▶ *Interface binding* binds the EIS service connection information to the adapter service interface.
- ▶ *Method binding* binds the relationship between the operation of the adapter service interface and the interaction or event of the adapter.
- ▶ *Data binding* binds the service input and output data type (SCA business graph) to the data used in the adapter for interaction (WBIRRecord). The specified Java class performs the conversion from a business graph to the native format of the resource adapter, which in this case is a WBIRRecord. JCA adapters use a `RecordHolderDataBinding` as described by the Enterprise Metadata Discovery specification. Refer to the *Enterprise Metadata Discovery specification* document for further detail, available at this Web site:

<ftp://www6.software.ibm.com/software/developer/library/j-emd/EnterpriseMetadataDiscoverySpecification.pdf>

Example 4-4 on page 81 shows an EIS import file.

Example 4-4 Example of an EIS import file

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:import xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:eis="http://www.ibm.com/xmlns/prod/websphere/scdl/eis/6.0.0"
xmlns:ns1="http://RedMaintenanceModule/interfaces/RedMaintenanceOutboundInterface"
xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/6.0.0"
xmlns:wSDL="http://www.ibm.com/xmlns/prod/websphere/scdl/wSDL/6.0.0"
displayName="RedMaintenanceOutboundInterface"
name="RedMaintenanceOutboundInterface">
  <interfaces>
    <interface xsi:type="wSDL:WSDLPortType"
portType="ns1:RedMaintenanceOutboundInterface">
      <method name="createApartment"/>
    </interface>
  </interfaces>
  <esbBinding xsi:type="eis:EISImportBinding"
dataBindingType="com.ibm.j2ca.extension.emd.runtime.WBIDataBindingImpl">
    <resourceAdapter name="RedMaintenanceModuleApp.RedMaintenance"
type="com.ibm.itso.sab511.RMResourceAdapter"/>
    <connection type="com.ibm.itso.sab511.outbound.RMManagedConnectionFactory"
interactionType="com.ibm.itso.sab511.outbound.RMInteractionSpec">
      <authentication resAuthAlias="widNode/Red_Alias"/>
    </connection>
    <methodBinding method="createApartment">
      <interaction>
        <properties>
          <functionName>Create</functionName>
        </properties>
      </interaction>
    </methodBinding>
    <methodBinding method="deleteApartment">
      <interaction>
        <properties>
          <functionName>Delete</functionName>
        </properties>
      </interaction>
    </methodBinding>
    <methodBinding method="updateApartment">
      <interaction>
        <properties>
          <functionName>Update</functionName>
        </properties>
      </interaction>
    </methodBinding>
    <methodBinding method="retrieveApartment">
      <interaction>
        <properties>
          <functionName>Retrieve</functionName>
        </properties>
      </interaction>
    </methodBinding>
  </esbBinding>
</import>
```

```
</properties>
</interaction>
</methodBinding>
</esbBinding>
</scdl:import>
```

4.6 Adapter configuration properties

It is likely that there are configuration properties that need to be defined in the adapter. For example, the EIS host name and port number, and an EIS user name and EIS password. These properties can be added to the adapter's deployment descriptor. The advantage of setting properties in the deployment descriptor instead of hard-coding the properties in the adapter code gives greater flexibility by allowing those properties to be set during adapter deployment.

If the properties were added to the `<resourceadapter>` section of the deployment descriptor, the corresponding Java bean properties need to be create in the adapter class that implements the interface, `javax.resource.spi.ResourceAdapter`.

If the properties were added to the `<outbound-resourceadapter>` section of the adapter deployment descriptor, the corresponding Java bean properties must be added to the adapter classes that implement the interfaces, `javax.resource.spi.ManagedConnectionFactory` and `javax.resource.spi.ConnectionRequestInfo`.

Note: The WebSphere Adapter Toolkit automatically creates the Java bean properties in the corresponding adapter classes when you add properties using the adapter deployment descriptor editor.

4.7 Application sign-on

Typically connecting to an EIS requires some sort of authentication. The process of authenticating an adapter connection to the EIS is referred to as sign-on. Sign-on can be done either in container-managed mode or component-managed mode. You can set the appropriate type of sign-on mode in the adapter deployment descriptor:

- ▶ `<res-auth>Container</res-auth>`
- ▶ `<res-auth>Component</res-auth>`

As the name implies, container-managed sign-on means the JCA container is responsible for providing sign-on credentials. The `javax.security.auth.Subject` is passed to the adapter from the JCA container for the sign-on to the EIS.

In component-managed sign-on mode, the adapter client passes the user name and password information to the adapter through the implementation class of the `javax.resource.cci.ConnectionSpec` interface.

Some EIS resources support reauthentication. Reauthentication allows changing the current security context of an existing physical EIS connection. If re-authentication is supported, you can set the `<reauthentication-support>` element of the adapter deployment descriptor to `true`:

```
<reauthentication-support>true</reauthentication-support>
```

See the WebSphere Adapter Toolkit user guide for more information about application sign-on.

4.8 Connection management

According to the JCA specification, adapters can run in managed and nonmanaged environments. In a managed environment, the application server has the responsibility to manage connection pooling, security, and transaction. In a nonmanaged environment, the client is responsible to manage these tasks. In this book, we only discuss developing an adapter to run in a managed environment. In section “System contracts” on page 38, the connection management contract was discussed and the connection process was illustrated.

Using SCA, the adapter client running on WebSphere Process Server does not need to get a handle to the physical EIS connection. Instead, the adapter client gets a handle to the SCA adapter service by calling the `locateService` method of the `ServiceManager` instance. The adapter service instance exposes the available operations of the adapter using the adapter service interface.

The EIS service import does the following:

- ▶ Binds the EIS service connection information to adapter service interface.
- ▶ Binds the service interface methods to adapter operations.
- ▶ Binds the input/output data type (business graph) from the adapter client to the adapter data type (WBIRRecord).

With a handle to the adapter service, the adapter client can invoke available operations specified in the adapter service interface. The SCA service runtime handles the adapter connection factory lookup and gets a connection to the EIS on behalf of the adapter service client.

The following interfaces need to be implemented by the adapter to comply with the adapter side of the JCA contract for outbound connection management:

- ▶ javax.resource.spi.ConnectionFactory
- ▶ javax.resource.spi.Connection
- ▶ javax.resource.spi.ManagedConnection
- ▶ javax.resource.spi.ManagedConnectionFactory
- ▶ javax.resource.cci.ConnectionSpec

4.9 Outbound interaction

An adapter is used to provide a standard interface on one end while it connects to an EIS-specific interface on the other end. This allows application components to interact with various EIS resources using a standard application programming model through resource adapters.

Using the standard JCA application programming model to interact with an EIS, the application component does the following:

1. Uses the JNDI service to look up a connection factory for the adapter and requests a connection from the connection factory.
2. Uses the Connection object to create an Interaction object.
3. Creates an InteractionSpec object and specifies the values required for execution.
4. Creates the Record object used to pass data into the EIS.
5. Executes the function through the Interaction object. The function execution returns data to the client through a Record object.
6. Closes the Interaction.
7. Closes the Connection.

Example 4-5 illustrates these steps.

Example 4-5 Sample code to show interaction with an EIS

```
WBIConnection conn;  
WBIRecord input;  
WBIRecord output;  
...  
Interaction interaction = conn.createInteraction();  
WBIInteractionSpec interactionSpec = new WBIInteractionSpec();  
interactionSpec.setFunctionName(WBIInteractionSpec.CREATE);  
output = interaction.execute(interactionSpec, input);  
...
```

With SCA, the adapter's client interface is the adapter's service interface. The adapter client invokes operations that are available from this interface.

Using information from the SCA import, the SCA runtime maps the interface operation to the adapter's operation. This includes setting the function name property in the `WBInteractionSpec` instance, or any other `InteractionSpec` property that may be required by the resource adapter.

Once it receives a connection instance to the EIS, the SCA runtime creates an `Interaction` instance and calls the `execute` method of the `Interaction` instance.

The implementation of the `execute` method does the following:

1. Extracts the function name from the `WBInteractionSpec` instance.
2. Converts the input `WBIRRecord` object to an EIS-specific object.
3. Calls the EIS function that maps to the retrieved function name.
4. Passes the EIS-specific object as an argument to the EIS function call.
5. Retrieves the result from the EIS function call and converts it to a `WBIRRecord` object.
6. Returns the `WBIRRecord` object.

Upon receiving the `WBIRRecord` object, the SCA runtime binds it to a business graph by requesting the data binding to perform the conversion. The data binding information is specified in the adapter service import file.

The SCA runtime then delivers the business object to the calling service component.

Note: There are two versions of the `execute` method in the `Interaction` interface:

- ▶ `public Record execute (InteractionSpec ispec, Record in)`
- ▶ `public boolean execute (InteractionSpec ispec, Record in, Record out)`

You can specify which `execute` method is called by SCA runtime in the class that extends `WBIRResourceAdapterMetadata`. Implement the following methods in this class and return the appropriate boolean value:

- ▶ `public boolean supportsExecuteWithInputAndOutputRecord()`
- ▶ `public boolean supportsExecuteWithInputRecordOnly()`

The following are interfaces that need to be implemented by the adapter to comply with the adapter side of the JCA contract for outbound interaction:

- ▶ `javax.resource.spi.Interaction`
- ▶ `javax.resource.spi.InteractionSpec`

4.10 Standard outbound operations and processing

The previous section described how the adapter service client can interact with the EIS through the `execute` method of the `Interaction` object. It described how the `execute` method gets the EIS function name from the `InteractionSpec` object that was passed into the `execute` method. This function name tells you which EIS function the adapter should invoke. EIS operations can range from Create, Retrieve, Update, and Delete (CRUD) operations for database application to operations that are completely unique to the customer's EIS instance. All adapters should make an effort to support the standard operations described in this section. If an EIS does not support an operation, then it is not required. The following describes the processing of the standard operations.

4.10.1 ApplyChanges operation

This is a catch-all operation that enables an adapter client to send any business graph that has the verb Create, Update, or Delete and process it accordingly. This operation saves effort and simplifies mapping. For after-image business objects, this operation looks at the top-level verb in the business graph and calls the corresponding Create, Update, or Delete operation. For delta business objects, this operation invoke the appropriate delta processing logic.

4.10.2 After-Image Create operation

The aim of the Create operation is to create a new entity in the EIS that matches the data and structure of the incoming business object. The business object returned by this operation is a mirror image that reflect the newly created entity in the EIS.

High level processing steps for this operation

Processing starts from the top-level business object:

1. Create an entity in the EIS with the data from the input business object
2. If the EIS does not generate its own primary key (or keys), insert the key values from the incoming business object into the appropriate key column (or columns) of the EIS entity.

3. Update the output business object to reflect the values of the newly created EIS entity; this includes any EIS-generated key values or properties marked as having potential side-effects.
4. Recursively create the EIS entities corresponding to the first-level child business objects, and continue recursively creating all child business objects at all subsequent levels in the business object hierarchy.

Interpretation of ChangedSummary

When interpreting ChangedSummary in the coming business graph:

- ▶ Business object level change summary details can be ignored for this operation. All business objects in the hierarchy should be created in the EIS
- ▶ Property-level change summary details should be used to determine whether a given property is to be set, blanked/cleared or simply ignored during processing.

Create operation return value

Return a copy of the incoming business object. The output business object should be modified only to reflect newly created key values and other side-effects. ChangeSummary logging and values should be kept as they are.

Error handling for Create operation

DuplicateRecordException is thrown if EIS already contains an entity with the same key values as a business object to be created.

InvalidRequestException is thrown if input to operation is not supported:

- ▶ If ChangeSummary is provided but required contained children are not marked as created (per strict conventions)
- ▶ Key values are specified in the input business object but EIS only supports auto-creation
- ▶ Key values are not specified in the input business object but EIS requires them
- ▶ Any business object is marked in the ChangeSummary as deleted (assertion optional)
- ▶ Top-level verb, if provided, is not Create (assertion optional)

EISSystemException is thrown if EIS reports any unrecoverable errors

4.10.3 After-Image Update operation

This operation modify an entity in the EIS in a way that this entity and any children entity match the data and structure exactly as the incoming business object. This operation requires explicit comparison of the incoming business object and the entity in the EIS system. ChangeSummary, if provided, should not be used alone since it does not reflect the child object deletions.

High level processing

The high level processing steps for this operation are:

1. Compare the existing business object in the EIS with the input business object and Create, Update, or Delete entities as needed to match the input.
 - 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 are not contained in the business object, are deleted from the application.
2. Update the output business object to reflect the modified EIS entities. This includes any EIS-generated key values or properties marked as having potential side-effects.

Interpretation of ChangedSummary

When interpreting the ChangedSummary in the coming business graph, the following to conditions are true:

- ▶ Business-object-level change summary details can be ignored for this operation. Use aforementioned comparison of the input business object to the existing entity in the EIS to determine appropriate changes.
- ▶ Property-level change summary details should be used to determine whether a given property is to be set, blanked or cleared, or simply ignored during processing.

Update operation return value

These are the two operations you can perform:

- ▶ Recommended

Create a copy of the input business object, enable ChangeSummary if not already, and then update ChangeSummary to reflect changes made to the EIS by the adapter including key values for newly created objects and side-effects. This includes removing deleted objects from the data graph and marking them as deleted in the ChangeSummary. Change the top-level verb

in the output business object to `UpdateWithDelete` to reflect the fact that the deleted child objects are now (should be) accurately reflected in the return object.

- ▶ **Acceptable**

Create a copy of the input business object and update it with any newly created key values or side-effects.

Error handling for Update operation

The Update operation includes all exceptions thrown by the create and delete operations, plus:

- ▶ `RecordNotFoundException` is thrown if EIS does not contain an entity with the same key values as a business object to be updated.
- ▶ `EISSystemException` is thrown if EIS reports any unrecoverable errors.

4.10.4 After-Image UpdateWithDelete operation

This is a special form of the Update operation that is intended to provide better performance. It always requires a `ChangeSummary` that includes business object-level creations and deletions. This enables the adapter to perform operations without the overhead of retrieving the existing entities from the EIS and doing comparisons since the `ChangeSummary` indicates what needs to be done. If `ChangeSummary` is empty, then the adapter does not take any action on the request.

Interpretation of ChangedSummary

When interpreting the `ChangedSummary` in a business graph, the following is true:

- ▶ Business objects marked as created in the `ChangeSummary` should be created in the EIS according to the logic described under the Create operation.
- ▶ Business objects marked as deleted in the `ChangeSummary` should be removed from the EIS according to the logic described under the Delete operation.
- ▶ Business objects without business object-level changes but with property-level changes should be updated in the EIS.
- ▶ In all cases:
 - Only properties marked as set in the `ChangeSummary` should be used to populate the EIS entities.

- For properties marked as ignore, do not modify the corresponding property in the EIS entity.
- For properties marked as blank, follow blank semantics.
- ▶ Business objects without BO-level or property-level changes should be ignored. This means if the entire ChangeSummary is empty, the adapter should not take any action.

UpdateWithDelete operation return value

Create a copy of the input data object and then update the ChangeSummary to reflect changes made to the EIS by the adapter, including key values for newly created objects and side-effects.

Error handling for UpdateWithDelete operation

This operation has the same error-handling as the Update operation.

4.10.5 After-Image Delete operation

This operation remove an existing entity from the EIS and any contained child entities.

High-level processing steps for this operation

The high-level steps are:

1. Perform a recursive retrieve on the input business object to get all data in the EIS that is associated with the top-level business object.
2. Perform a recursive delete on the entities represented by the input business object, starting from the lowest-level entities and ascending to the top-level entity. Noncontained entities should be left intact, although any relationships to deleted objects should be removed if explicitly defined in the EIS.

Note: Adapters should also delete any and all contained children, irrespective of whether they are reflected in the input business object. For example, if just a top-level business object is provided with keys and no children, the adapter should still check for contained children in the EIS and delete them.

Interpretation of ChangedSummary

When interpreting the ChangedSummary in the coming business graph, the following statements are true:

- ▶ If ChangeSummary is included with input business object, then the adapter should delete all business objects marked deleted in the change summary portion of the business graph and their contained children.
- ▶ If ChangeSummary is not included, then the adapter should delete all business objects specified in the data portion of business graph and their contained children.

Delete operation return value

Because the deletion of an entity in the EIS only requires a return that indicates the success (or failure) of the operation, the goal is to convey this with as little overhead as possible. Ideally, this would simply be a Boolean flag, but because that is not supported, the recommend approach is to return an empty business graph of the same type as the input business object. If this is not possible, it is acceptable to simply return a copy of the input business object (with ChangeSummary logging and values preserved).

The top-level verb of the input business object should be maintained in the output in either case.

Error-handling for Delete

RecordNotFoundException is thrown if EIS does not contain an entity with the same key values as a business object to be deleted.

InvalidRequestException is thrown if input to operation is not supported:

1. If ChangeSummary is provided and any contained children are not marked as deleted (per strict conventions).
2. Any BO is marked in the ChangeSummary as created (assertion optional).

EISSystemException is thrown if EIS reports any unrecoverable errors.

4.10.6 Retrieve operation

The retrieve operation rebuilds the complete business object hierarchy that matches exactly the database state of the application entity.

Nonkey values can be used as criteria. The Retrieve operation accepts either an after-image or delta business object. The comparison in either case is by equality only.

The request business object might contain any of the following:

- ▶ A top-level business object but no child objects, even though the business object definition includes children
- ▶ A business object that contains the top-level business object and some of its defined children
- ▶ A complete hierarchical business object containing all child business objects

High-level processing steps for this operation

The steps are:

1. Create a copy of the input business object.
2. Clear the ChangeSummary and disable ChangeSummary logging.
3. Recursively retrieve records using EIS values provided in input business object and populate output business object:
 - The adapter should populate in the output business object any children that are defined in the EIS but incomplete in the input business object.
 - The adapter should add to the output business object any children that are defined in the EIS but for which no child instance is included in the input business object.
4. If change summary logging is enabled in the input business object, re-enable it in the response business object.

Error-handling for Retrieve

`RecordNotFoundException` is thrown if any populated business object in input object does not exist in the EIS.

`MultipleMatchingRecordsException` is thrown if more than one record matches the input criteria.

`EISSystemException` is thrown if EIS reports any unrecoverable errors.

4.10.7 RetrieveAll operation

`RetrieveAll` operation returns a *batch* of records that match the values provided in the request business object. In such a batch retrieval, the adapter returns a collection of business objects by means of a top-level *container* business object as shown Figure 4-2 on page 93. The operation always returns a result set irrespective of how many (if any) matches are found. If no values are provided in the request business object, all objects of the given type should be returned (such as wildcards for all properties).

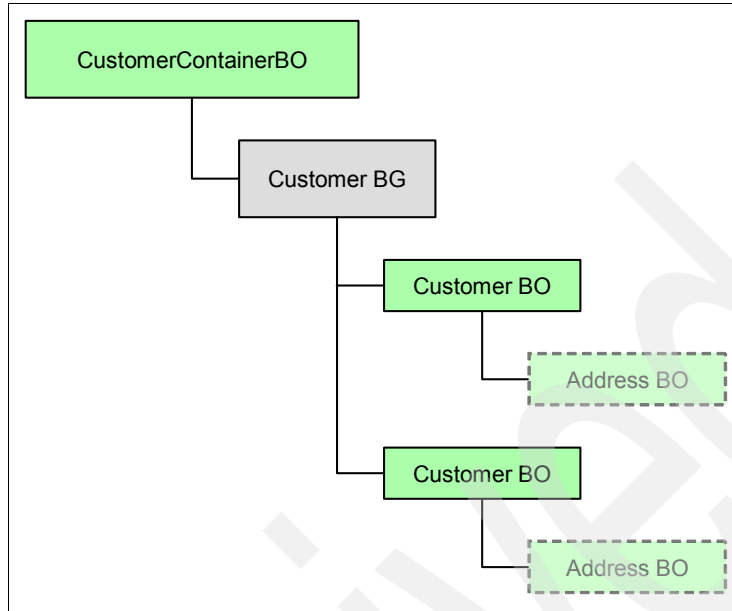


Figure 4-2 Collection of business objects

Adapter clients that support batch results must be capable of recognizing this top-level container and understand to iterate through the child objects which represent the results of the query. Any individual BG in the container, then, can be extracted by the client and delivered through the rest of the system the same as single business object. This obviously requires the creation of additional business object *container* structure definitions by the user or during metadata import for each business object type that the user intends to query in batch.

This approach was adopted over JDBC-style `ResultSet` support as described in the JCA specification. However, if an EIS provides native `ResultSet` support or it makes sense, then adapter developers are encouraged to implement the CCI `ResultSet` interfaces to provide customers with a high-performance alternative to the generic batch retrieval approach described in this document.

Note: The use of the operation name *RetrieveAll* rather than *RetrieveByContent* as was used with WebSphere Business Integration (existing) adapters is done to distinguish this operation as a new and clear standard. Support for *RetrieveByContent* was inconsistent across existing adapters: some would retrieve a single object and return special error codes if there were more objects that matched, while other adapters would create special containers to return multiple values.

High-level processing steps for this operation

In general, the steps are the same as 4.10.6, “Retrieve operation” on page 91.

Adapters should check property *MaxRecords* in the *WBIInteractionSpec* instance to determine the maximum number of records to return in order to avoid out-of-memory issues. The difference between *Retrieve* and *RetrieveAll*, is that *Retrieve* is intended to return a single, unique business object that meets user-defined criteria, while *RetrieveAll* returns multiple, matching business objects. For example, use *Retrieve* to find Customer where *id=abc123* and *RetrieveAll* to find all Customers where *state=NY*.

RetrieveAll operation return value

For *RetrieveAll*, the adapter performs a query and retrieves a result set of all objects that match a given set of values. The output object is a container that holds an 0..n objects of the same type as the input object.

Error-handling for RetrieveAll

The *RecordNotFoundException* is thrown if any populated business object in input object does not exist in the EIS.

MatchesExceededLimitException throw if the number of hits in the EIS exceeds the value of *MaxRecords* defined in the interaction specification. Property *MatchCount* contains the actual number of hits that the adapter had in the EIS so that users can either increase their limit or refine their search appropriately.

EISSystemException is thrown if EIS reports any unrecoverable errors.

4.10.8 Custom operations

Adapters may support custom operations that enable users more robust means of interrogating or modifying the EIS. Examples of such operations include *Execute* to execute a stored procedure or script, or *Lock* to lock an entity in the EIS.

For such operations, it is recommended that the operation implementations accept both after-image and delta business objects and simply use the values provided in the data portion of the business graph. If a delta is provided and not enough information is included to perform the request operation, the adapter should attempt to retrieve the necessary data from the EIS application, if possible, or otherwise throw an *InvalidRequestException* expanding on the missing content.

It is assumed that the *ChangeSummary* and the top-level verb can be and is ignored for all custom operations. However as with other operations, both the

existing ChangeSummary and top-level verb of the input BO should be copied into the output business object to ensure consistency (unless it does not make sense, as with operation Retrieve which clears the ChangeSummary before returning.)

4.11 Command Pattern

In the previous section, we provide an overview of standard adapter operations and how adapters must process these operations. As you can see, processing these operations can be quite complex. For example, to process an after-image Update operation, the adapter must have logic to retrieve the existing entity hierarchy from the EIS, compare it to the incoming business object, and invoke the operations (Update, Delete or Create) necessary to make the EIS entity structure match the incoming business object.

WebSphere Adapter Foundation Classes provides a Command Pattern to abstract this functionality into generic logic. We can leverage this generic logic offered by the Command Pattern so we do not have to start from the beginning every time we develop an adapter.

Note: The Command Pattern is suitable for any EIS that supports typical CRUD operations. A brief overview of the Command Pattern is presented here. For details on the Command Pattern, see the *WebSphere Adapter Toolkit User Guide*. The WebSphere Adapter Toolkit is available from the developerWorks® Web site. See:

<http://www-128.ibm.com/developerworks>

The idea behind the Command Pattern is to break down a business object hierarchy into a hierarchy of small subcommands. The hierarchy of subcommands is fed into an Interpreter. The Interpreter traverses the incoming hierarchy of subcommands and executes the EIS-specific sub-commands. This relieves you from having to traverse the incoming business object hierarchy tree when processing adapter operations. It makes possible for the adapter developer to deal with adapter operations on a single-tier level, regardless of the height of the incoming business object hierarchy tree. The Command Pattern greatly simplifies adapter development.

The Command Pattern offers the following advantages:

- The adapter operation's EIS-specific code is separated from generic operations.

- ▶ It gives the adapter the ability to operate in phantom mode. If operations must be tried, the interpreter can easily be turned off, with the contents of the command hierarchy dumped to a file instead.
- ▶ The only code that the adapter developer needs to write is the EIS-specific operation code. The comparator and interpreter code would be common components.
- ▶ As common components can be used across adapters, the after image of the update process can be rigorously defined, making different adapters work more consistently.

The Command Pattern consist of the following main components:

- ▶ **Command Manager**

The Command Manager breaks down the incoming business object hierarchy into nodes, then creates sub-commands that can deal with a node (see Figure 4-3 on page 97). For an Update operation, the Command Manager also compares the incoming business object hierarchy and the EIS object hierarchy before creating the appropriate commands for the nodes (Figure 4-4 on page 98).

- ▶ **Command Interpreter**

The Command Interpreter takes the command hierarchy produced by the Command Manager and executes the sub-commands according to the order of execution for the operations. At the end of the Command Interpreter's execution, the EIS is updated and the resulting business object is returned.

- ▶ **Subcommands**

The adapter developer implements an EIS-specific subcommand for each outbound operation, for example:

- CreatCommand
- DeleteCommand
- UpdateCommand
- RetrieveCommand

These subcommands are used by the Command Manager for the nodes in the command hierarchy.

4.11.1 Command Pattern processing snapshot objects

The Command Manager creates a command structure based on the incoming business object that consists of nodes with subcommands. The command structure is then fed into the Interpreter which traverses the command structure tree and executes each individual subcommand. Each subcommand has an

EIS-specific implementation that invokes the operation on the EIS. See Figure 4-3.

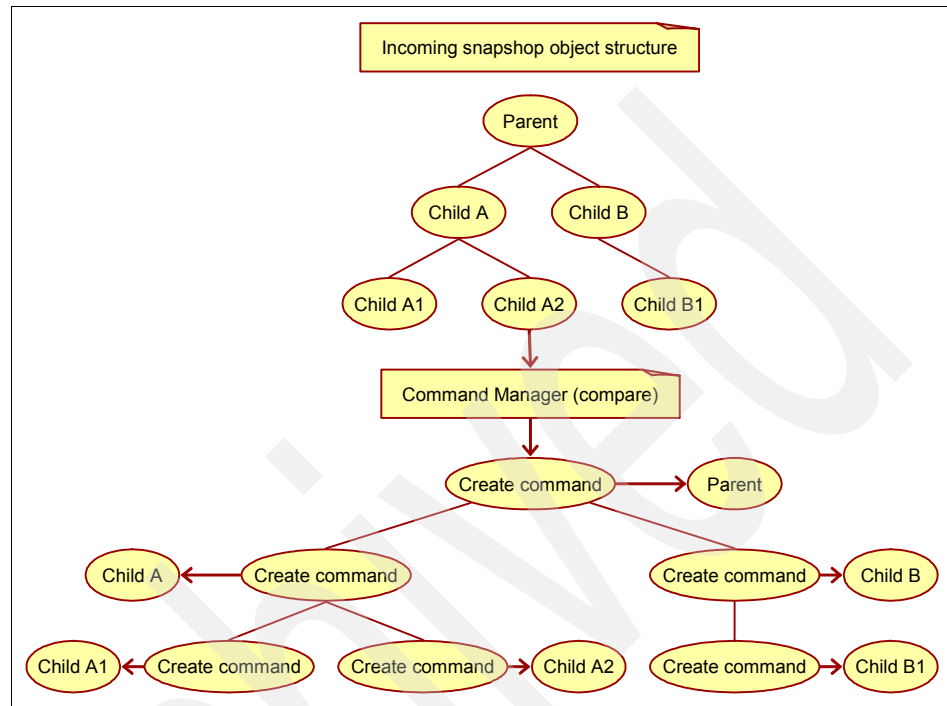


Figure 4-3 Command Manager command structure for after-image CREATE

For a snapshot of Update, the Command Manager compares the incoming business hierarchy with the retrieve object from the EIS. It then builds the command structure that changes the entity in the EIS to match the incoming business object structure. See Figure 4-4 on page 98.

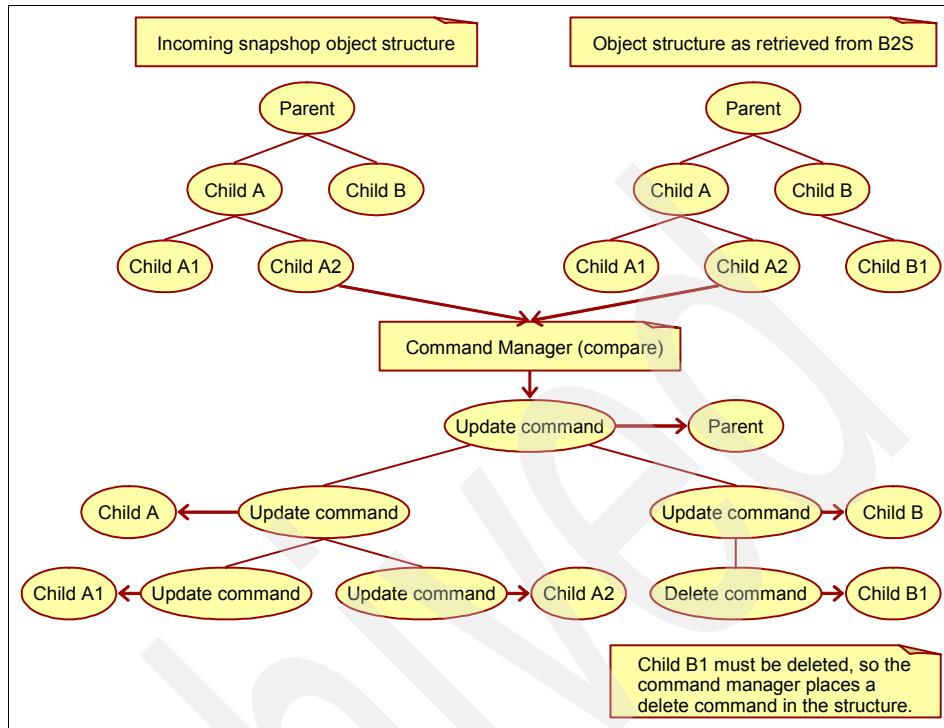


Figure 4-4 Command Manager command structure for after-image Update

4.11.2 Command Pattern processing delta objects

The same idea is used to process delta objects. The Command Manager reads the change summary of the business object instead of doing a retrieve and compare. **NO_OPERATION** commands are used for those unchanged parents. See Figure 4-5 on page 99.

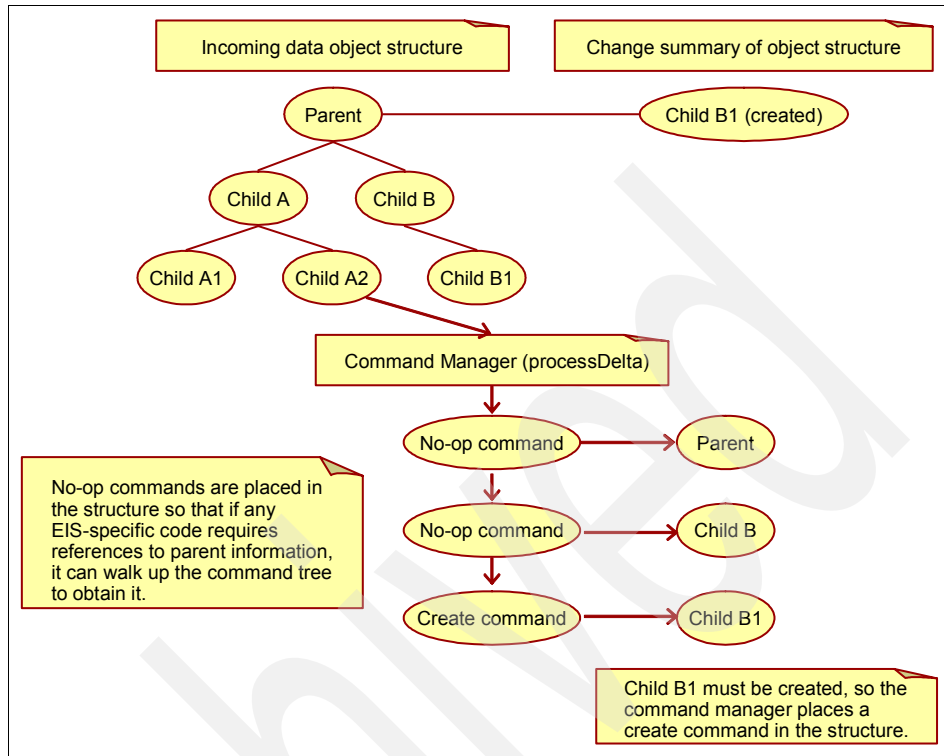


Figure 4-5 Command Manager command structure for delta CREATE

4.11.3 How to use the Command Pattern

To use the Command Pattern, follow these steps:

1. Implement a Command for each of the supported adapter CRUD operations. For example:
 - a. Create a CreateCommand class that extends Command class of the Command Pattern. In the execute method, implement logic to create a new entity in the EIS. The implementation should be concerned only with the logic needed to create a single entity that matches the incoming business object. The Command Pattern creates the children entities based on the incoming business object definition.

Example 4-6 on page 100 shows a sample implementation of the execute method. You must include logic to link the current object to the parent object if the incoming business object has multiple cardinality relationship.

Example 4-6 Sample CreateCommand class's execute method.

```
DataObject execute(DataObject obj){
    EisRepresentation eis = EisAPI.insert(toEisFormat(obj));
    return (toDataObject(eis));
}
```

- b. Create a DeleteCommand class that extends Command class of the Command Pattern. In the execute method, implement logic to delete the entity in the EIS. If the EIS delete API does not automatically delete children entities, implement the logic here to perform a recursive delete.
 - c. Create an UpdateCommand class that extends Command class of the Command Pattern. In the execute method, implement logic to update the EIS entity that matches the incoming business object. If the incoming business object has children entities, the Command Pattern automatically handles the recursive updates.
 - d. Create a RetrieveCommand class that extends the Command class of the Command Pattern. In the execute method, implement logic to retrieve recursively the corresponding EIS objects for the incoming business object. Unlike the CreateCommand and UpdateCommand, the Command Pattern does not perform the recursion for you here.
2. Implement a CommandFactory that creates an instance of the adapter operation command for the Command Manager. See Example 4-7.

Example 4-7 Sample Command Factory implementation class

```
public class RMCommandFactoryImpl implements CommandFactory {
    ...
    public com.ibm.j2ca.extension.commandpattern.Command createCommand(
        java.lang.String functionName, commonj.sdo.DataObject dataObject)
        throws javax.resource.ResourceException {
        RMBaseCommand command = null;
        if (functionName.equals(NodeLevelOperations.CREATE_NODE)) {
            command = new RMCreateCommand(dataObject);
        } else if (functionName.equals(NodeLevelOperations.DELETE_NODE)) {
            command = new RMDeleteCommand(dataObject);
        } else if (functionName.equals(NodeLevelOperations.UPDATE_NODE)) {
            command = new RMUpdateCommand(dataObject);
        } else if (functionName.equals(NodeLevelOperations.RETRIEVE_STRUCTURE))
        {
            command = new RMRetrieveCommand(dataObject);
        } else if (functionName.equals(NodeLevelOperations.RETRIEVE_ALL)) {
            command = new RMRetrieveAllCommand(dataObject);
        } else {
            command = new RMBaseCommand(dataObject);
        }
    }
}
```

```

        command.setObjectConverter(objectConverter);
        command.setObjectNaming(objectNaming);

        if (functionName == NodeLevelOperations.DELETE_NODE) {
            command.setExecutionOrder(Command.BEFORE_PARENT);
        } else {
            command.setExecutionOrder(Command.AFTER_PARENT);
        }
        return command;
    }
    ...
}

```

3. Implement the execute method for the Interaction class:

- a. Call CommandManager to produce the command structure.
- b. Call the interpreter to execute the subcommands in the command structure.

See Example 4-8.

Example 4-8 Sample implementation of the execution method in Interaction class.

```

public javax.resource.cci.Record execute(
    javax.resource.cci.InteractionSpec ispec,
    javax.resource.cci.Record inRecord)
    throws javax.resource.ResourceException {
    WBIInteractionSpec interactionSpec = (WBIInteractionSpec) ispec;
    String functionName = interactionSpec.getFunctionName();
    Command topLevelCommand = commandManager.produceCommands(
        (WBIRecord) inRecord, functionName);
    DataObject returnDataObject;
    returnDataObject = interpreter.execute(topLevelCommand);

    WBIRecord outRecord = new WBIRecord();
    if (functionName == WBIInteractionSpec.RETRIEVE_ALL_OP) {
        outRecord.setDataObject(returnDataObject);
    } else {
        outRecord.setDataObject(returnDataObject.getContainer());
    }
    logger.traceMethodExit(RMInteraction.class.getName(), "execute()");
    return outRecord;
}

```

4.12 Transaction support

Using adapters, application components running on a application server can access and update data in multiple Enterprise Information Systems. Quite often such data access and updates must be performed in a transactional manner. A *transaction* is a unit of work that has the following Atomicity, Consistency, Isolation, Durability (ACID) properties:

- ▶ Atomicity

Logical operations are grouped as an atomic unit. Either all the operations must succeed or all must fail when executed as an atomic unit.

- ▶ Consistency

At the end of a successful transaction, the system must be in a consistent state.

- ▶ Isolation

When transactions occur concurrently, each transaction should appear to execute independently of other transactions. The net effect of executing a set of transactions concurrently should be the same as that of running them serially.

- ▶ Durability

The effect of the transaction should be persistent.

When an application component running in an application server executes a transaction to multiple Enterprise Information Systems, the transaction manager implemented in the application server takes the responsibility of coordinating transactions across multiple Enterprise Information System.

Under the JCA specification, adapters can support two types of transactions: local transactions and XA transactions.

- ▶ Local transaction

A local transaction is managed internally by the adapter and does not require coordination by an external manager. A local transaction is also called a *one-phase commit protocol*. It allows a given client to demarcate the start and end of transactional operations with the EIS application alone. See Figure 4-6 on page 103.

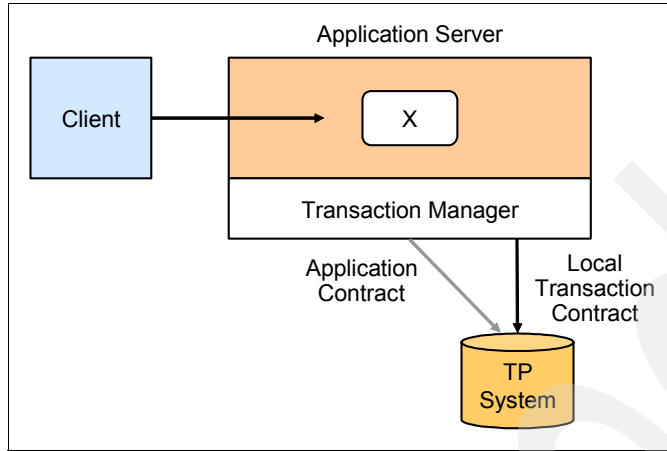


Figure 4-6 Local Transaction

► XA Transaction

An XA transaction can span multiple heterogeneous Enterprise Information Systems. It requires transaction coordination by an external transaction manager. A transaction manager uses a two-phase commit protocol to manage an XA transaction that spans multiple Enterprise Information Systems. See Figure 4-7.

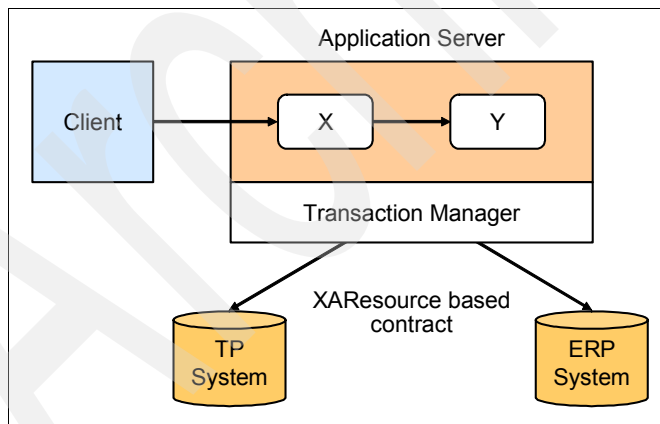


Figure 4-7 XA Transaction

Note: While an adapter can expose support for either or both protocols, the underlying EIS must ultimately provide the support. You would not normally attempt to implement XA support in your adapter if your underlying EIS application inherently lacked transaction support.

When you have determined that your EIS supports transactions, you must make the following modifications to your adapter to expose the support:

1. Update the adapter deployment descriptor property `TransactionSupport` with one of the following values:

- `NoTransaction`
- `LocalTransaction`
- `XATransaction`

Any adapter that supports XA should also provide support for local transactions. If your adapter does support XA, mark support as `XATransaction` support but also implement the local transaction features described here.

2. Update Your Adapter-Specific Constructor of `WBIResourceAdapterMetadata` class to return true for the following method:

```
public boolean supportsLocalTransactionDemarcation()
```

3. Override the following methods:

- `WBIManagedConnection.getLocalTransaction()`
- `WBIManagedConnection.getXAResource()`. Only if XA is supported.

4. Wrap the `LocalTransaction` and `XAResource` instances returned by these methods with a `WBILocalTransactionWrapper` or `WBIXATransactionWrapper` instance, respectively. These wrappers provide extended diagnostics for troubleshooting and also help adapters determine whether or not to autocommit requests. According to the JCA 1.5 specification, a resource adapter must autocommit transactions when being used outside the context of a transaction.

To help the managed connection determine if it is involved in a transaction, these wrappers act as thin delegation layers that monitor the sequence of calls to determine whether a transaction is active. At the beginning of a transaction, the wrappers call method `setEnlistedInATransaction(true)` on the `WBIManagedConnection` instance. Upon commit or rollback, the wrappers set this same property to false. By then, checking the status of the transaction via method `isEnlistedInTransaction` on the super class, a `WBIResourceAdapter` subclass can quickly determine whether it should be autocommitting transactions or not when modifying the EIS. See Example 4-9 on page 105.

Example 4-9 Example of an XA-enabled adapter implementation

```
public class FooManagedConnection extends WBIManagedConnection {
    // just get the XAResource from your EIS and return the wrapper
    public XAResource getXAResource() {
        XAResource eisXAResource = this.eisXAConnection.getXAResource();
        XAResource wrapper = new WBIXATransactionWrapper(eisXAResource, this);
        return wrapper;
    }

    // here's an example of a potentially transacted call on the EIS. Point
    // is that adapter should always check whether it's enlisted in a
    // container-managed transaction or whether it should handle transaction
    // on its own
    private void updateRecord(int id, int value) {
        if (!this.isEnlistedInTransaction())
            this.eisConnection.beginTransaction();
        eisConnection.updateRecord(id, value);
        if (!this.isEnlistedInTransaction())
            this.eisConnection.commitTransaction();
    }
}
```

Inbound processing and events

This chapter describes briefly the fundamentals of inbound processing and events management in the resource adapter development. In this chapter we discuss the following:

- ▶ Inbound processing overview
- ▶ The JCA message inflow contract contract
- ▶ Event management
- ▶ Inbound processing in the Service Component Architecture
- ▶ Inbound processing development steps

Note: This chapter provides a detailed theory of inbound processing. However, If you want a basic understanding of inbound processing, refer to Part 2, “Custom adapter development” on page 157, then an understanding of 5.1, “Inbound processing overview” on page 108 and 5.5, “Inbound processing development steps” on page 129 is necessary.

5.1 Inbound processing overview

In Chapter 4, “Outbound request processing” on page 73 we explain how an application component can interact with an Enterprise Information System (EIS) by invoking the adapter’s outbound operations. In addition to outbound communication with the EIS, the application component can also be interested in events that occurred on the EIS. For example, if the EIS is a database system, the application component might want to know when a new record is created in a database table. It can take appropriate actions then in response to this new EIS event.

These events can be delivered to the application component in two ways:

- ▶ The adapter polls the EIS, pulls the event from the EIS, and delivers it to the application component (Asynchronous inbound processing).
- ▶ The EIS pushes the event to the adapter for delivery to the application component (Synchronous inbound processing).

The J2EE Connector Architecture (JCA) 1.5 specification supports both synchronous and asynchronous inbound processing. In this book, we mainly focus on the Asynchronous/polling method to process inbound EIS events.

Using JCA adapters to handle inbound processing has the following advantages:

- ▶ The application components can delegate the inbound event processing to the resource adapter.
- ▶ The adapter converts the EIS event into a WebSphere Adapter business object. This business object is an extension of Service Data Objects (SDO) and is consumable by service components running on WebSphere Process Server. Service components do not need to know the EIS-specific data format.
- ▶ Multiple application components can *subscribe* to the adapter’s message endpoints and be notified of events that occurred on the EIS.

5.1.1 Inbound processing flow

Figure 5-1 shows how an EIS event flows from the EIS to the application component.

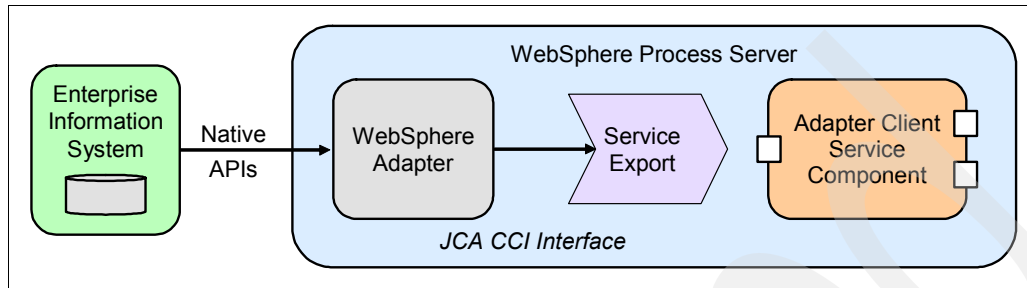


Figure 5-1 Inbound processing flow

The following steps describe asynchronous inbound processing flow:

1. An event occurs on the EIS. If the EIS is a database system, the event could be a record was added to a table, a record was updated or a record was deleted.
2. When a event occurs, a mechanism is needed to write the event to a event table or event store. If the EIS is a database system, database trigger is typically used to write a event entry into the event store.
3. The adapter can retrieve event information from an event store by polling the event store.
4. Clients interested in receiving events have to register with the adapter using the ActivationSpec of the adapter and the client endpoint has to implement the listener interfaces that the adapter defines.
5. The adapter delivers the events to these endpoints. During event processing, the adapter fetches the data from the EIS (referred to by the event), convert the data into a WebSphere Adapter business object and sends it to the listeners that suscribed to this event.

To implement the endpoint delivery and to interact with other services in Service Component Architecture (SCA), an EIS service export file provides the binding between the adapter and the client interface.

5.1.2 Inbound components

The Adapter Foundation Classes packaged in the WebSphere Adapter Toolkit standardizes the inbound message listener and the event delivery mechanism

for the different adapters. Figure 5-2 shows the components that enable the event management provided by the Adapter Foundation Classes.

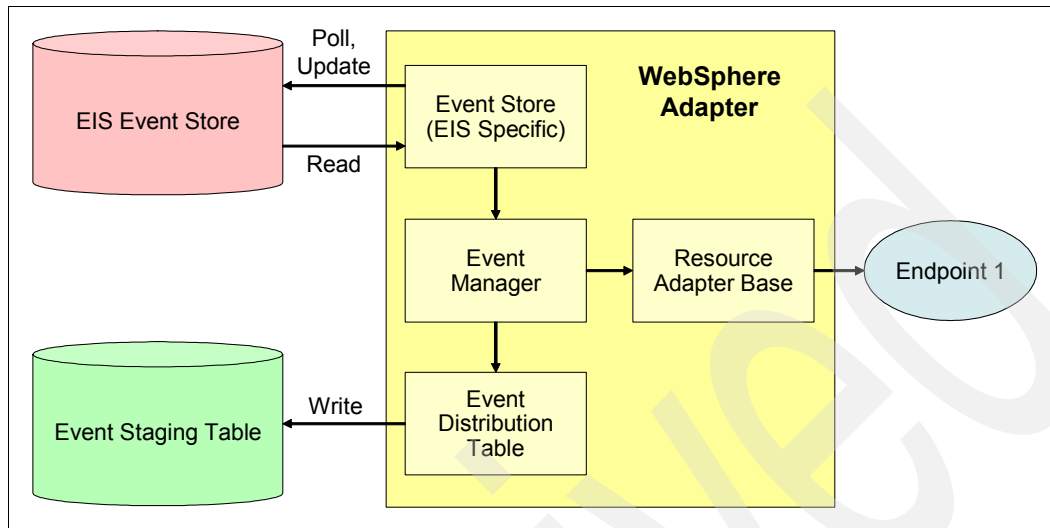


Figure 5-2 Inbound components

Components that enable the event management are:

- ▶ To enable polling, A event table sometimes called an *event store* need to be added to the EIS's database. This table is used to record events that must be processed by the adapter. When the adapter has successfully delivered the event to the endpoints, the corresponding event is removed from this table. It is important to note that not all EIS back-ends have an EIS store, for example, SAP EIS does not have an EIS Store and it lets the adapter manage the events.
- ▶ The *event staging table* is a special table managed by the WebSphere Adapter. It is used to ensure the *once and only once* delivery of events. The event staging table can be implemented using a persistent store like a database or can be kept in-memory. Using a persistent event staging table is preferred because it supports server failover.
- ▶ The *resource adapter base* classes take the events from the staging table and deliver them to the configured endpoints. These endpoints are the points the resource adapter uses to connect to other SCA artifacts. The EIS event store and event staging table are updated to indicate the status of the events within the adapter as the event is being processed within the adapter and the endpoint.

5.2 JCA message inflow contract

As described in the previous chapters of the book, the WebSphere Adapter technology fully compliant with JCA specification 1.5. If you use the Adapter Foundation Classes to develop a custom adapter, you only need to provide EIS specific implementations that extend the Adapter Foundation Classes. The Adapter Foundation Classes provides default implementations to satisfies the adapter side of the JCA contracts. This greatly speed up the development of a custom adapter.

This section gives an overview of the JCA Message Inflow system contract in order to provide a better understanding of the inbound processing. It provides the basis of inbound processing.

The message inflow JCA system contract allows the resource adapter to pass synchronous or asynchronous inbound messages to message endpoints on the application server. Message inflow is one of the key concepts of inbound processing because it defines all the necessary details that ensure successful communication between the adapter and the J2EE applications or SCA artifacts that consume the services provided by this adapter.

The main features of message inflow contract are:

- ▶ Allows message consumption without changing the Java 2 Platform, Enterprise Edition (J2EE) client programming model by implementing event delivery to message endpoints running on the J2EE application server.
- ▶ Defines a standard contract for plugging a message provider into the J2EE application server.
- ▶ Allows the delivery of different types and sizes of messages.
- ▶ Supports concurrent delivery of messages.

The actors that participate in the message inflow are shown in the Figure 5-3.

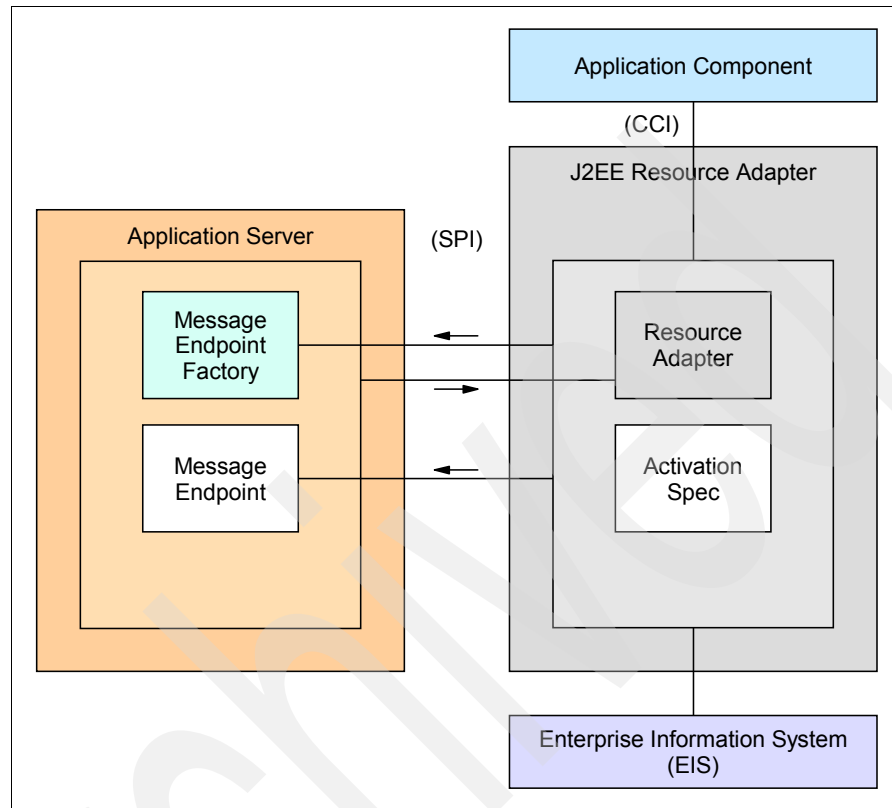


Figure 5-3 Message Inflow actors

The actors that participate in the message inflow are:

► **MessageEndpoint**

The message endpoint is the application deployed in the application server that is able to consume messages from the adapter. It can receive the messages synchronously (through Enterprise JavaBeans (EJB) invocation) or asynchronously (using message driven beans).

► **MessageEndpointFactory**

The message endpoint factory brings different message endpoint instances to the resource adapter for delivering messages serially or concurrently and can be used to check if a specific message endpoint is transacted or not.

► ResourceAdapter

The resource adapter basically supports the methods that perform activation and deactivation of message endpoints. It uses the message endpoint factory to obtain message endpoint instances. It also provides a list with all the endpoint message listener types it supports.

► ActivationSpec

This JavaBean class is provided by the resource adapter to each message listener type. It contains configurable properties used as the configuration information during the endpoint activation. The *ActivationSpec* class provides a validate method that can be overwritten in order to check the configuration information in the deployment stage.

The interaction between the actors of the message inflow is depicted in Figure 5-4.

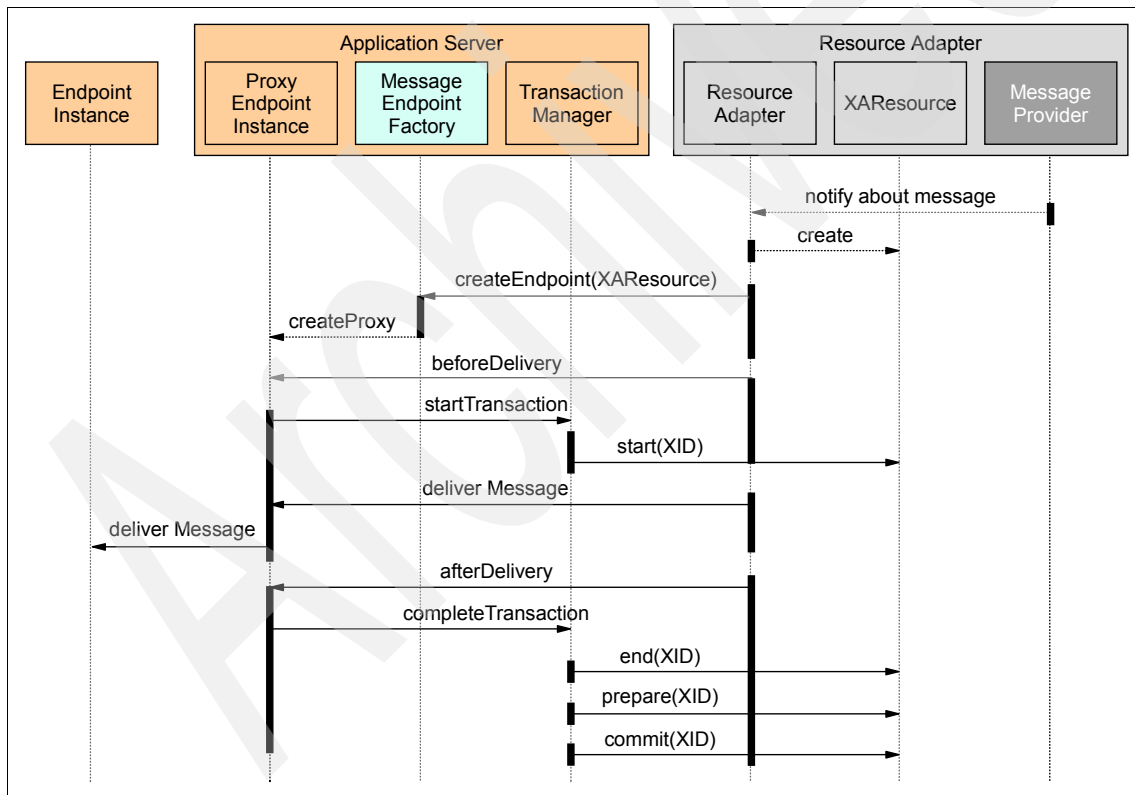


Figure 5-4 Message inflow sequence diagram

The message flow steps are:

1. The resource adapter is notified with the arrival of a new event.
2. The *MessageEndpointFactory* in the application server creates an endpoint proxy to the final endpoint. This endpoint proxy implements the resource adapter interface along with the *MessageEndpoint* interface and controls the transaction and parameters passing through the endpoint instance.
3. The event is delivered to the message endpoint inside a transaction context. It contains *beforeDelivery* and *afterDelivery* methods in order to provide some extra functionality.

The resource adapter must support multiple endpoints with similar activation configuration and it must treat them independently (sending a different copy of the event). The delivery of events can be serial or concurrent.

The application server handles redelivery of messages upon crash recovery if needed.

Note: See the JCA specification for more information about message inflow contract:

<http://java.sun.com/j2ee/connector/index.jsp>

5.3 Event management

To understand inbound processing we must first understand inbound event management. This section discusses the different aspects of event management:

- ▶ Event overview
- ▶ Event management process
- ▶ Event management classes
- ▶ Once and only once delivery
- ▶ Error handling in event management

5.3.1 Event overview

An *event* represents the creation, modification or deletion of an entity in the EIS. An *event store* is a persistent cache in the EIS where event records are saved until the adapter can process them. There are no standards to define the structure or content of an event record in the event store.

The basic attributes that an event must contain are presented in Table 5-1.

Table 5-1 Basic attributes of an event record in an EIS

Attribute	Description
Event ID	Each event requires an event ID for tracking purposes. As with any ID, this must be a unique identifier in the table.
Business Object Name	The event is always related with an entity type in an EIS. This attribute represents the name of the business objects the event refers to.
Business Key	This attribute represents the key value of the business object related to the event. This key is used later by the adapter when it is delivering the message to the endpoint.
Verb	Represents the operation that the event is triggering: Create, Update, Delete (One of the standard verb used by adapters).
Timestamp	The time at which the EIS generated the event.
Status	The status of the event. This information is used by the adapter to determine <i>new</i> versus <i>in process</i> events.

The event status must be compliant with the possible values described in Table 5-2.

Table 5-2 Possible event status values

Event Status	Description	Foundation Class Constant
New	The event is ready to be processed	NEWEVENT (0)
In Progress	The adapter is processing this given event. Note that an event in this state may or may not be delivered.	INPROGRESS (3)
Processed	The adapter successfully processed and delivered the event.	PROCESSED (1)
Failed	The adapter was unable to process this event due to one or more problems.	FAILED (-1)
Unsubscribed	The adapter processed the event but found no interested subscribers.	UNSUSCRIBED (2)

5.3.2 Event management process

In order to implement the event management functionality, the EIS application must guarantee the following:

- ▶ The event data must be stored in a persistent storage such as an event table.
- ▶ An event record in the event store should remain available in the event store until deleted by the adapter regardless of connection failure or time elapsed.
- ▶ The event store must allow the adapter to both identify and change the state of event records in the event store.
- ▶ While the event status values do not have to be exactly like those shown in Table above, but the set of status values must not change over time.

The general process of event management is shown in the Figure 5-5.

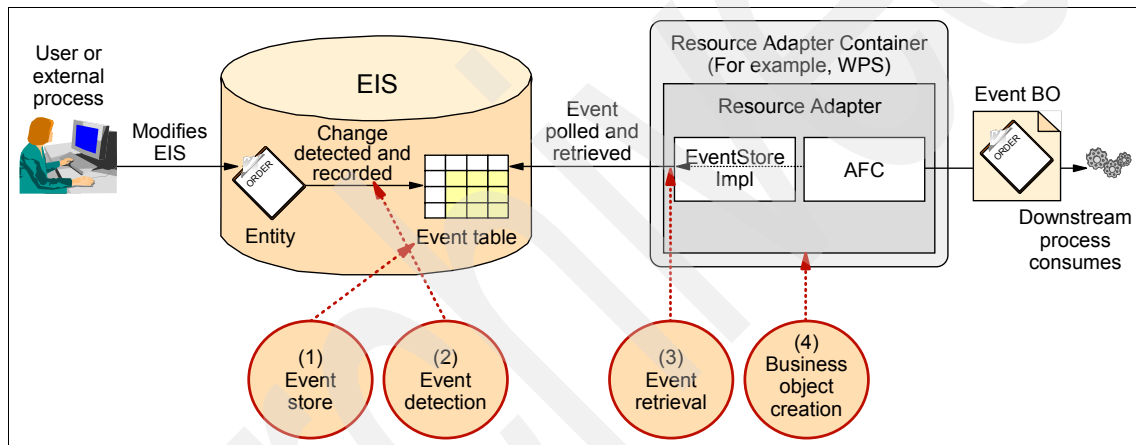


Figure 5-5 Event management

The primary events in the process are:

1. A user or external process modifies the EIS data.
2. An event detection mechanism implemented in the EIS detects changes of interest in the EIS and records them in the event store/table.
3. An event retrieval mechanism (event store interface) is implemented in the adapter that can detect or poll and retrieve events from the event store or table.
4. A data transformation mechanism is implemented in the adapter to convert the EIS event data to WebSphere Adapter business object. This conversion is necessary because service components in WebSphere Process Server can only consume SDOs.

The developer can implement the event management functionality using the Adapter Foundation Classes found in the WebSphere Adapter Developer Toolkit. The use of the adapter foundation classes for event management.

It is highly recommended that we use Adapter Foundation Classes for custom adapter development. Adapter Foundation Classes provide the following:

- ▶ Automatic tracking of the message endpoints during the event process
- ▶ Control of the polling for and delivery of the events
- ▶ Assurance of once and only once delivery
- ▶ Recovery of events if the adapter unexpectedly terminates

5.3.3 Event management classes

The information presented in this section describes the most important classes and interfaces included in the WebSphere Adapter Toolkit and related to inbound processing. An event management class diagram is shown first, and then the main classes are explained on a high-level.

Event management class diagram

The Figure 5-6 on page 118 shows a class diagram of event management for inbound processing with the most important classes and interfaces.

The developer can overwrite the following methods in order to get a specific implementation of the EventManager class:

- ▶ initialize

This method is called in the initialization process of the adapter. Its objective is to configure all the necessary elements in the EventManager class prior the event processing.

- ▶ pollForEvents

This method requests a number of events (based on a pollQuantity parameter) from the EIS event store and performs all the processing for them. The adapter developer can overwrite this method to get the EIS events from a different repository than the event store.

The resource adapter calls this method based on a timer task.

- ▶ getTarget

This method defines the method it passes to beforeDelivery on the targeted endpoint. The resource adapter developer can overwrite this method in order to call any method on the targeted interface. It is useful in certain circumstances, such as to invoke the same routine if two different business objects have been changed.

- ▶ addEndpointFactory

This method is called from the resource adapter in the endpointActivation method. It signals the EventManager that a new client is waiting for events.

- ▶ removeEndpointFactory

This method is called from the resource adapter in the endpointDeactivation method. It signals the EventManager that a client is not longer listening for events.

Event distribution

Together, EventDistributionResource and EventDistributionResourceDBImpl provide the functionality to operate a staging table. EventDistributionResource is the generic interface and EventDistributionResourceDBImpl is an implementation of this interface using an XA-compliant database. While, they are described here to help provide a further understanding of event processing, it is rare that an adapter developer needs to change the staging table implementation. Following are some of the methods in this class:

- ▶ storeEventForEndpoint

This method inserts a row into the staging table for a targeted client (endpoint).

- ▶ `getEventIdsForEndpoint`
This method retrieves all the eventIDs for a given client (endpoint).
- ▶ `storeEvents`
This method makes a block insert of events in the staging table, even for different endpoints.
- ▶ `getXaResource`
This method returns the XAResource representing the current database connection.
- ▶ `close`
This method closes all the prepared statements.
- ▶ `cloneConnection`
This method returns a new connection object with the same credentials as the current connection.

EventStore interface

An EventStore interface is supplied so that the EventManager can provide support for retrieval of events from the EIS event store. The EventStore interface is used to represent the event store and provides methods to retrieve and delete events from the underlying event store as well as methods to create connections to the event store. The resource adapter developer must provide an implementation of this interface.

The following is true of the EventStore interface:

- ▶ The implementation can hold either references to events or complete events.
- ▶ It should manage its own connection to the EIS (as opposed to outbound scenarios in which the JCA container manages the connection).
- ▶ The implementation should return events in the same order they appeared in the event table.

The main methods of this class are:

- ▶ `getObjectForEvent`
Retrieves the object from the EIS represented by the given event and returns an object that can be delivered to the message endpoint.
- ▶ `getEvents`
Returns an ArrayList of events with the given status. The events must be returned in order.

- ▶ `deleteEvent`
Removes an event given its event ID. May move the event to an archive store depending upon the implementation.
- ▶ `getSpecificEvent`
Retrieves a specific event from the EIS event store based on its ID. This method is used in recovery processes.
- ▶ `commitWork`
Commits the local transaction.
- ▶ `rollbackWork`
Rolls back the local transaction.

Important: The `EventManager` relies on the implementation of `getEvents` method to provide events in the order that are expected by the endpoint. This is needed in polling and recovery. The `EventManager` does not sort the events in any way.

Event class

The `Event` class represents the events that occur in the EIS and which are stores in the event store. This is a structural class and contains the following information:

- ▶ `eventId`
The unique identifier of the event.
- ▶ `eventKeys`
Keys used to identify an event for lookup in the EIS or to complete an event.
- ▶ `eventStatus`
Indicates event status.
- ▶ `eventTimestamp`
Represents the time the event was put in the event table.

All these attributes have their corresponding getter and setter methods. The adapter developer can extend this class in order to manage customized attributes.

5.3.4 Once and only once delivery

The event management foundation is based in the mechanism of once and only once delivery. This mechanism is provided by the adapter foundation classes in

order to guarantee no repudiation and no repetition of event delivery from the EIS to the message endpoints.

This mechanism uses a staging table as a temporary repository in which all the events coming from the EIS are stored and processed on a one-by-one basis and in a transacted environment. After an event is processed, it is deleted from the staging table and the EIS event store.

These considerations are described in the following four steps of event management:

1. A new event is added to the EIS event store and the status of the event is marked as New. During the poll time, the adapter fetches all the events marked New. In the case where there is no EIS event store, the event is pushed to the adapter by the backend EIS. Figure 5-7 illustrates this.

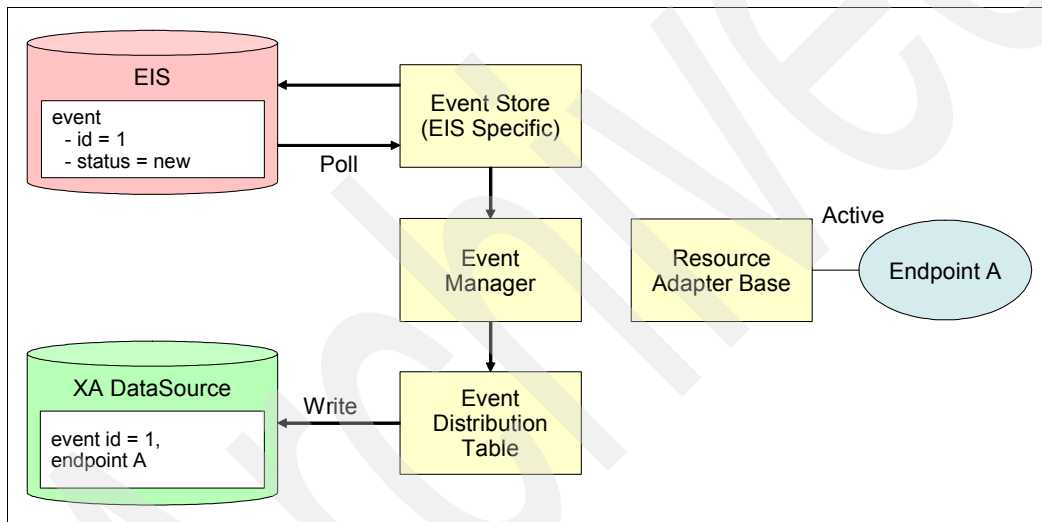


Figure 5-7 Event management: step 1

2. The event is saved in the event staging table within the adapter along with the event ID and the endpoint that is the registered listener of the event. In addition, the event is marked as In progress in the event store.

Figure 5-8 illustrates this.

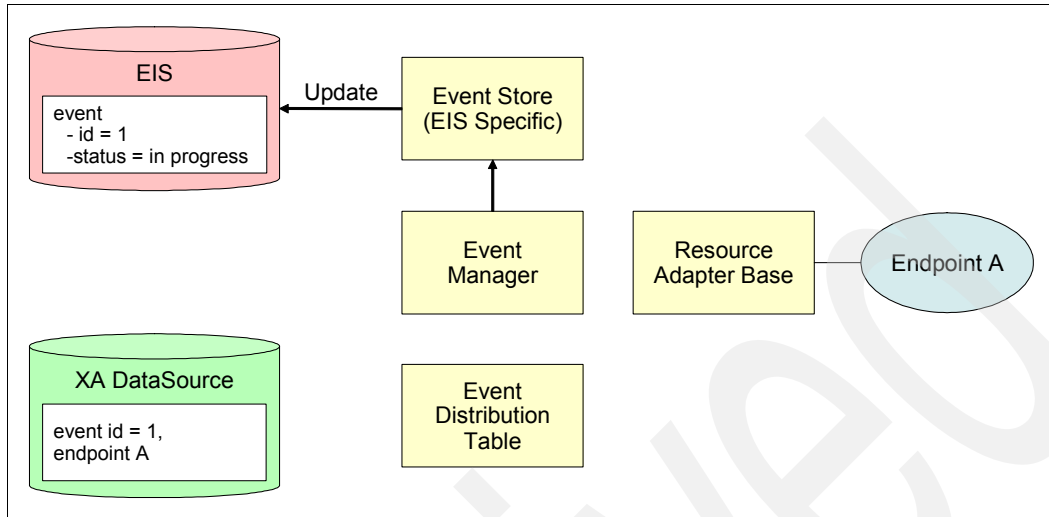


Figure 5-8 Event management: step 2

3. The event is published to the active endpoint and deleted from the staging table. The two actions might be part of an XA transaction if the endpoint supports transactions. The delivery of the event to the endpoint indicates the end of the transaction. See Figure 5-9.

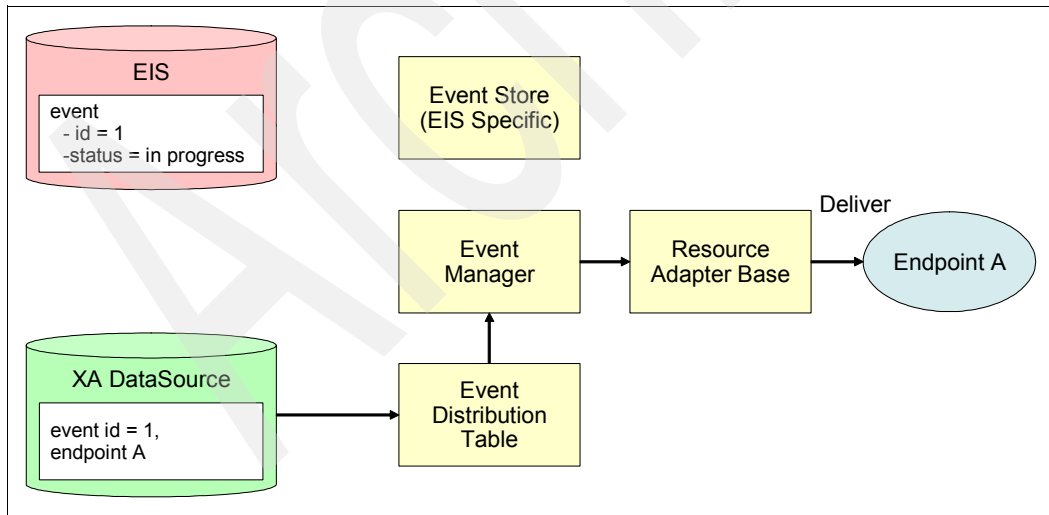


Figure 5-9 Event management: step 3

4. After the endpoint has received the event, the original event in the event store is marked for deletion. The event is now considered processed, as in Figure 5-10.

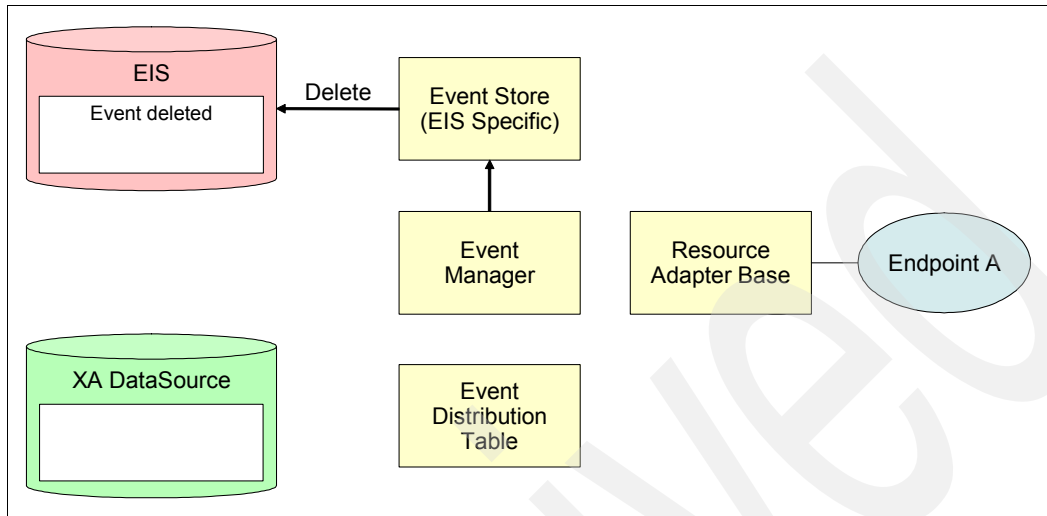


Figure 5-10 Event management: step 4

5.3.5 Error handling in event management

Inbound processing can present errors, most of them related to runtime configuration and interaction with third party systems. These errors can be present during the event processing, during the endpoint delivery, and during the event store capture.

Errors during event processing

In order to ensure the once and only once delivery, it is not possible to assume that the event manager works continuously. When failures occur and the event manager is interrupted on a restart, it looks at the event status in the EIS event store and in the staging table to see what steps need to be performed on the event. These steps (1-4) are described in 5.3.4, “Once and only once delivery” on page 121.

Table 5-3 shows how once and only once event delivery is assured, even if the event manager is interrupted.

Table 5-3 Events error handling

Event State		Interpretation	Action by Event Manager
In EIS Event Store	In Staging Table		
Event marked as new	No record of event	This event is a new event being detected for the first time.	Perform steps 1-4
Event marked as new	One or more records corresponding to this event exist	The event manager was previously interrupted after it detected and copied the event to the staging table, but before it could update the event as in-progress.	Perform steps 2-4
Event marked as in-progress	One or more records corresponding to this event exist	The event manager was in the process of delivering the event when it was interrupted.	Perform steps 3-4
Event marked as in-progress	No record of event	The event manager successfully delivered the event, but it was interrupted before it could delete the event from the event store.	Perform step 4

Errors during endpoint delivery

If the endpoint throws an exception during the delivery, the event manager has to stop delivering events to that endpoint. The timer task for polling has to be stopped as well. What happens to the other events retrieved in the poll cycle depends on the delivery type:

- ▶ If the delivery type is ORDERED, the remaining events polled in that cycle cannot be delivered until the event with the error is processed.
- ▶ If the delivery type is UNORDERED, the event manager attempts to deliver the remaining events in the current poll cycle.

When the failing endpoint is taken offline and reactivated the event in which the error occurred must be redelivered and then normal delivery must continue.

Finally, if the endpoint is transactional, and the transaction rolls back, the behavior of the event manager is the same as though the endpoint threw an exception.

Errors during event store capture

If the event store throws an exception during the process of full retrieval of the event object, the event manager must change the status of that event in the event table to `ERROR_PROCESSING_EVENT`. If this happens and the delivery type is `ORDERED`, the polling task must stop until the endpoint is reactivated. If the delivery type is `UNORDERED`, the polling task can continue.

5.4 Service Component Architecture and inbound processing

The adapters described in this book are intended to work with the new Service Component Architecture on WebSphere Process Server. In this section we explain the different components that are involved to deliver the inbound event to the target service component running on WebSphere Process Server.

5.4.1 Adapter Clients

Adapter clients are application components running on the application server. In WebSphere Process Server an adapter client can be an SCA service component wired to the adapter through an EIS service export.

5.4.2 Service input and output data

When the adapter detects a new event in the event store, it uses the business object name and key in the event entry to retrieve the native EIS business object. It then converts this native business object into a WebSphere Adapter business object then to a business graph before sending it to the message endpoints.

In order for the application component to interact with the EIS through the adapter, the business object and business graph definitions must be created. They can be created manually, or if the adapter implements Enterprise Meta Discovery (EMD) they can be mined automatically from the underlying EIS using the Enterprise Service Discovery wizard in WebSphere Integration Developer.

5.4.3 Adapter service interface

An adapter service interface is defined by a Web Services Description Language (WSDL) port type. Arguments and return values are described using the business object definition Extensible Markup Language (XML) schema. An adapter exposes its available outbound operations to the client through its service interface. The adapter inbound interfaces are made available to the client

SCA components. The interface defines the type of business object that is delivered by the adapter.

An adapter service interface can be created using WebSphere Integration Developer's interface editor or if the adapter implements EMD, it can be created automatically using the Enterprise Service Discovery wizard.

Example 5-1 shows a sample of the interface WSDL file.

Example 5-1 Example of an inbound service interface file

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:ApartmentBG="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/
apartmentbg"
xmlns:MaintenanceBG="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenan
ce/maintenancebg"
xmlns:PartOrderBG="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/
partorderbg"
xmlns:intf="http://RedMaintenanceModule/RedMaintenanceInboundInterface"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
name="RedMaintenanceInboundInterface.wsdl"
targetNamespace="http://RedMaintenanceModule/RedMaintenanceInboundInterface">
  <types>
    <xsd:schema
xmlns:tns="http://RedMaintenanceModule/RedMaintenanceInboundInterface"
xmlns:xsd1="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/apartme
ntbg"
xmlns:xsd2="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/mainten
ancebg"
xmlns:xsd3="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/partord
erbg" elementFormDefault="qualified"
targetNamespace="http://RedMaintenanceModule/RedMaintenanceInboundInterface"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import
namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/apartmen
tbg" schemaLocation="ApartmentBG.xsd"/>
      <xsd:import
namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/partorde
rbg" schemaLocation="PartOrderBG.xsd"/>
      <xsd:import
namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/maintena
ncebg" schemaLocation="MaintenanceBG.xsd"/>
      <xsd:element name="emitCreateAfterImageMaintenance">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="emitCreateAfterImageMaintenanceInput"
type="xsd2:MaintenanceBG"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </types>
  </definitions>
```

```

        </xsd:complexType>
    </xsd:element>
    <xsd:element name="emitUpdateAfterImageMaintenance">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="emitUpdateAfterImageMaintenanceInput"
type="xsd2:MaintenanceBG"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
</types>
<message name="emitCreateAfterImageMaintenanceRequest">
    <part element="intf:emitCreateAfterImageMaintenance"
name="emitCreateAfterImageMaintenanceRequestPart"/>
</message>
<message name="emitUpdateAfterImageMaintenanceRequest">
    <part element="intf:emitUpdateAfterImageMaintenance"
name="emitUpdateAfterImageMaintenanceRequestPart"/>
</message>
<portType name="RedMaintenanceInboundInterface">
    <operation name="emitCreateAfterImageMaintenance">
        <input message="intf:emitCreateAfterImageMaintenanceRequest"
name="emitCreateAfterImageMaintenanceRequest"/>
    </operation>
    <operation name="emitUpdateAfterImageMaintenance">
        <input message="intf:emitUpdateAfterImageMaintenanceRequest"
name="emitUpdateAfterImageMaintenanceRequest"/>
    </operation>
</portType>
</definitions>

```

5.4.4 EIS service export

The EIS service export is an SCA export that allows the service components inside the SCA modules to receive information from an adapter client representing an EIS service defined outside of the SCA module. The EIS export binding binds the external EIS service to the SCA module. It specifies the mapping between the definitions of the inbound message endpoints that the adapter provides with the operation that must be implemented in a service component to subscribe to the events generated through these endpoints. The EIS export binding is done at three levels:

- ▶ Interface binding
- ▶ Method binding
- ▶ Data binding

An Example of an EIS import file is shown in Example 5-2 on page 129.

Example 5-2 Example of an EIS export file

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:export xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:eis="http://www.ibm.com/xmlns/prod/websphere/scdl/eis/6.0.0"
xmlns:ns1="http://RedMaintenanceModule/RedMaintenanceInboundInterface"
xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/6.0.0"
xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/6.0.0"
displayName="RedMaintenanceInboundInterface"
name="RedMaintenanceInboundInterface" target="SCATestComponent">
  <interfaces>
    <interface xsi:type="wsdl:WSDLPortType"
portType="ns1:RedMaintenanceInboundInterface">
      <method name="emitCreateAfterImageMaintenance"/>
      <method name="emitUpdateAfterImageMaintenance"/>
    </interface>
  </interfaces>
  <esbBinding xsi:type="eis:EISExportBinding"
dataBindingType="com.ibm.j2ca.extension.emd.runtime.WBIDataBindingImpl">
    <resourceAdapter name="RedMaintenanceModuleApp.RedMaintenance"
type="com.ibm.itso.sab511.RMResourceAdapter">
      <properties/>
    </resourceAdapter>
    <connection type="com.ibm.itso.sab511.inbound.RMActivationSpec"
selectorType="com.ibm.j2ca.extension.emd.runtime.WBIFunctionSelectorImpl">
      <properties>

<BONamespace>http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance</BONam
espace>
      </properties>
    </connection>
    <methodBinding method="emitCreateAfterImageMaintenance"
nativeMethod="emitCreateAfterImageMaintenance"/>
    <methodBinding method="emitUpdateAfterImageMaintenance"
nativeMethod="emitUpdateAfterImageMaintenance"/>
  </esbBinding>
</scdl:export>
```

5.5 Inbound processing development steps

The following steps are required to implement inbound processing functionality in a custom adapter:

- ▶ Define the inbound properties
- ▶ Implement the connection with the EIS event store

- ▶ Analyze the need for a custom Event class
- ▶ Implement the custom EventStore class
- ▶ Modify the ActivationSpec subclass
- ▶ Modify the ResourceAdapter subclass
- ▶ Implement an event store or not

5.5.1 Define the inbound properties

The first step the adapter developer should take is to define the properties the inbound processing needs to perform properly.

The ActivationSpec class contains a set of standard configurable properties used in the endpoint activation process. These properties are listed in Table 5-4. Change these values to the required value and ensure any default values used fit the needs of the resource adapter.

Table 5-4 ActivationSpec reserved and standard properties

Name	Type	Required?	Description
PollPeriod	Integer	Yes	<p>This property represents the rate (in milliseconds) at which the resource adapter polls the EIS event store for new inbound events.</p> <p>Its value must be an integer with value greater or equal than zero (0). If set to zero, the adapter does not wait between cycles.</p> <p>If for some reason the poll cycle takes longer than expected to complete, the next cycle is executed immediately and then the fixed rate is established.</p>
PollQuantity	Integer	Yes	This property is used to determine the number of events to deliver to each endpoint in each poll cycle.
AutoCreateEDT	Boolean	No	Flag that indicates whether the adapter should create the Event Distribution Table (EDT or staging table) automatically if it does not exist already.
EDTDriverName	String	No	<p>Name of the XA database driver to use to connect to the EDT for inbound events.</p> <p>Despite it is not a required value, if there is no value present, the Event Manager is unable to perform recovery.</p>
EDTDatabaseName	String	No	Name of the database (inside the RDBMS) where the EDT belongs.

Name	Type	Required?	Description
EDTSchemaName	String	No	Name of the of the schema (inside the selected database) where the EDT is configured.
EDTTableName	String	No	Name of the table in the properly schema.
EDTUserName	String	No	Database user to manipulate the EDT records.
EDTUserPassword	String	No	Password of the database user.
EDTURL	String	No	JDBC connection URL to get the database.
EDTServerName	String	No	Name of the server where EDT is running.
EDTPortNumber	String	No	TCP port number of the listening process of the selected database.
DeliveryType	String	No	This attribute determines the order in which the events are published. The only allowed values are: ORDERED (One at a time) UNORDERED (All at once).

If a property is required that is not covered by the standard set of inbound properties, it can be included in a list of custom properties in the ActivationSpec. As a suggestion, the adapter developer can make a list similar to the shown in Table 5-5. This list is used later in the implementation phase (11.2, “Identify application properties and operations” on page 253).

Table 5-5 Inbound custom properties

Inbound Properties	Values
Property 1	Property value 1

5.5.2 Implement the connection with the EIS event store

In inbound processing, the resource adapter is responsible to provide its own implementation of the connection management to the EIS.

In order to do that, the resource adapter must wrap the mechanism implemented in the specific EIS APIs to get and release connections from it. As every EIS could have different parameters to get and release connections, the resource adapter only has to take care of the definition of the following two methods:

► getConnection

This method is responsible for receiving all the parameters specified by the EIS API, then it is responsible for the establishment of the connection as the

documentation of the EIS suggests. The result of the invocation must be an object or interface representing the specific connection to the EIS.

- ▶ `releaseConnection`

This method is responsible for the release of a connection obtained in the previous method, and according to the EIS specific instructions.

5.5.3 Analyze the need for a custom Event class

The Adapter Foundation Classes provide a default Event class (see “Event class” on page 121). However, it is possible that the event entity provided by the EIS contains more information attributes than the JCA Event class provides for. In this case, a custom Event subclass must be created in order to properly synchronize the information between the EIS event store and the staging event table.

5.5.4 Implement the custom EventStore class

The adapter foundation classes provide an EventStore interface (see “EventStore interface” on page 120). The adapter developer must provide an implementation class that takes the following into consideration:

- ▶ Filtering

Filtering refers to the ability of the EIS to query events based on filters. The filtering functionality is implemented in the method called `implementsFiltering`. If the EIS event store can filter events by type in the `getEvents` call, this method should return *true*. Otherwise it should return *false*.

- ▶ Get events (polling)

Polling for events is the main functionality the adapter developer has to provide in the EventStore implementation. This functionality is implemented in the method called `getEvents`. The following is a list of the tasks to implement:

- Receive the `pollQuantity`, `status`, and `eventTypeFilter` parameters.
- Get the EIS connection.
- Get the events of the EIS event table according the `pollQuantity` number, `status` and `eventTypeFilter` parameters.
- For each of these events:
 - Create an adapter event from the EIS event data.
 - Put this adapter event in an `ArrayList`.
- Sort the list of events.
- Return the ordered list.

► Delete an event

The event store must provide a way to delete an event from the EIS event store. This functionality is implemented in the method `deleteEvent`. This method is called during the event management process, and the adapter developer has to implement the algorithm with the following steps:

- a. Create an EIS Event from the Adapter Event Data.
- b. Execute a delete operation of this Event in the EIS.

► Update event status

The event store must also provide the mechanism to update an EIS event with the information the resource adapter event, mostly for synchronization purposes. This implementation must be made in the method called `updateEventStatus`. The adapter developer has to implement the algorithm with the following steps:

- a. Create an EIS Event from the Adapter Event Data and the new status value.
- b. Execute an Update operation of this Event in the EIS.

► Get a specific event

The event store provides the mechanism to update the information of an adapter event with the information that is saved in the EIS event store, primarily for synchronization purposes. The adapter developer has to implement the following steps in the method called `getSpecificEvent`:

- a. Look for an event in the EIS with the `eventId` provided as parameter.
- b. Create an Adapter Event instance from the data received from the EIS.
- c. Return the Adapter Event instance.

► Get a business object from an event

The `getObjectForEvent` method is another crucial method that must be implemented. The objective of this method is provide a way to get the business object graph from the EIS entity information related with some event. The adapter developer has to implement the algorithm with the following steps:

- a. Receive an instance of adapter event class.
- b. Create a dummy business graph (BG) business object instance.
- c. Create a dummy specific data business object.
- d. Fill the data business object with the appropriate data from the EIS (using the `RetrieveCommand` of outbound processing).
- e. Put the data object instance inside the business graph instance.
- f. Set the BG Object verb to the event's verb.

- g. Create a WBIRecord with the BG Object instance.
- h. Return the WBIRecord.
- Transactional considerations
You have to analyze if the EIS application supports transactional event processing or not. If the answer is true or false, the adapter developer must implement the following methods with the properly functionality:
 - isTransactional
 - rollbackWork
 - commitWork

5.5.5 Modify the ActivationSpec subclass

Normally the functionality provided by the Adapter Foundation Classes is not enough to satisfy the needs of a custom implementation because the custom resource adapter usually needs to deal with some customized properties. In these cases, the resource adapter must implement a subclass of the ActivationSpec class in order to handle all the custom inbound properties. The ActivationSpec class uses the Observer design pattern in order to dynamically search and manage these properties.

The review of the customized ActivationSpec subclass consists of these steps:

1. Ensure that the class extends the `com.ibm.j2ca.base.WBIActivationSpec` class.
2. Ensure that the class has already coded the custom properties (and the proper getter and setter methods) defined in the previous step.

5.5.6 Modify the ResourceAdapter subclass

To put all the pieces together, you must modify the ResourceAdapter subclass in order to overwrite the method called `createEventStore`.

The implementation of this method has to return the appropriated EventStore class instance.

5.5.7 Implement an event store or not

In most cases, the EIS has implemented an event store, and this store can be represented as a database table, a file in a filesystem or other persistence object. If this is the case, the way to get the events from the EIS is by defining a

customized EventStore class and implementing the required methods, as listed above. Provide access to the event store could involve any of the following:

- ▶ Direct access to the EIS event table using Java Database Connectivity (JDBC).
- ▶ The use of some proprietary EIS API.
- ▶ If the event store is implemented in a file or another kind of repository, the adapter developer should provide the functionality in order to handle this type of EventStore.

If the EIS does not have an implementation of the event store, the adapter developer must choose one of the following paths:

- ▶ If it is possible, the adapter developer can create an event table and can populate it using database triggers or by modifying the underlying EIS functionality.
- ▶ If the event store cannot be created or implemented, the adapter developer has to determine if the EIS pushes the events and then overwrite the EventManager class in order to avoid the use of an EventStore in the process of getting the events directly from the EIS.

The theory of Enterprise Service Discovery

In this chapter, the theory and architecture of Enterprise Metadata Discovery (EMD) for resource adapters is discussed.

Note: To better understand EMD, it is recommended that you read the white paper *Enterprise Metadata Discovery* published by IBM and BEA. This paper is available at this Web site:

<http://www-128.ibm.com/developerworks/library/specification/j-emd/>

The following topics are discussed in this chapter:

- ▶ EMD related terms and definitions
- ▶ Overview of Enterprise Metadata Discovery
- ▶ Architecture of the Enterprise Metadata Discovery
- ▶ Roles and responsibilities in Enterprise Metadata Discovery development
- ▶ Generating SCA service interfaces using Enterprise Service Discovery Tool
- ▶ Resource Adapter EMD component development process

6.1 EMD definitions

We first define some common Enterprise Metadata Discovery (EMD) related terms before we discuss it:

- ▶ Enterprise Information System (EIS) Metadata

Metadata is data used to describe other data. Metadata can also refer to the definition of data. EIS metadata is used to describe the data stored in the EIS systems. For example, table names and column names could be part of the EIS metadata.
- ▶ WebSphere Integration Developer EMD tools

WID is an Integration Developer Environment. It has built in plug-in of EMD Tools that implements the EAI tooling side of the EMD specification.
- ▶ Enterprise Metadata Discovery (EMD)

EMD is an adapter component that implements the adapter side of the EMD specification.
- ▶ Enterprise Service Discovery (ESD)

ESD consist of two components, the EMD implementation in the Enterprise Application Integration (EAI) tooling and the EMD implementation in the adapter. After an adapter is imported into the EAI tooling environment such as WebSphere Integration Developer (WID), the user can run the ESD to automatically generate the SCA artifacts and business objects needed for an integration solution that runs on WebSphere Process Server.
- ▶ Application-specific information (ASI)

ASI is the metadata that describes a business object, its verbs, and fields. Annotations in the business object definition XML Schema Definition (XSD) files is used to encapsulate ASI metadata.

6.2 Enterprise Metadata Discovery overview

A JCA resource adapter is a system-level software driver for connecting to a specific Enterprise Information System (EIS). The J2EE JCA specification defines how the resource adapter plugs into an application server and how the J2EE applications call the EIS services through the resource adapter by using Common Client Interface (CCI) APIs. By complying the specification, the resource adapter has a standard interfaces to interact with J2EE application server and J2EE applications.

However, to connect to EIS, the integration developers or users need to know the following:

- ▶ What services are exposed by EIS.
- ▶ Generate business objects that represent data and metadata in the EIS. These business objects must be in form consumable by the application server. For example, we create WebSphere adapter business objects that extends SDO for passing data to and from the adapter to WebSphere Process Server.

Manually creating information is a painful job, considering that some data structures in EIS are really complex. In order to simplify the process of creating service descriptions provided by EIS, it is a natural extension for the resource adapter to have the capability to browse and generate metadata and service information from the EIS. To provide this capability, a white paper *Enterprise Metadata Discovery (EMD) Specification* was created by IBM and BEA to define a standard contract between the resource adapter and Enterprise Application Integration (EAI) tooling frameworks. The paper is available here:

<ftp://www6.software.ibm.com/software/developer/library/j-emd/EnterpriseMetadataDiscoverySpecification.pdf>

The EMD specification introduces a metadata discovery and import model for resource adapters and EAI tooling. This model allows resource adapters to easily plug into an EAI tooling (For IBM, the tool is included in WebSphere Integration Developer v6). The tooling provides the user interface, generating process and common utilities for all resource adapters. The resource adapter developers need to implement specific interfaces that are called by EAI tools to connect and retrieve metadata from the EIS. In the rest part of this chapter, detail architecture and roles for this model are introduced.

6.3 Architecture

Figure 6-1 on page 140 shows the basic architecture of resource adapters with the feature of Enterprise Metadata Discovery. It describes the relationships between Application Server, tooling and resource adapter. The advantage of this architecture is that tooling can be common for all resource adapters that support EMD. This allows the separation of concerns, meaning tools developer can focus on tools development while adapter developer can focus on the interaction of the adapter with the EIS.

In the section, we first explain details about each component in Figure 6-1. The relationship between each pair is also described.

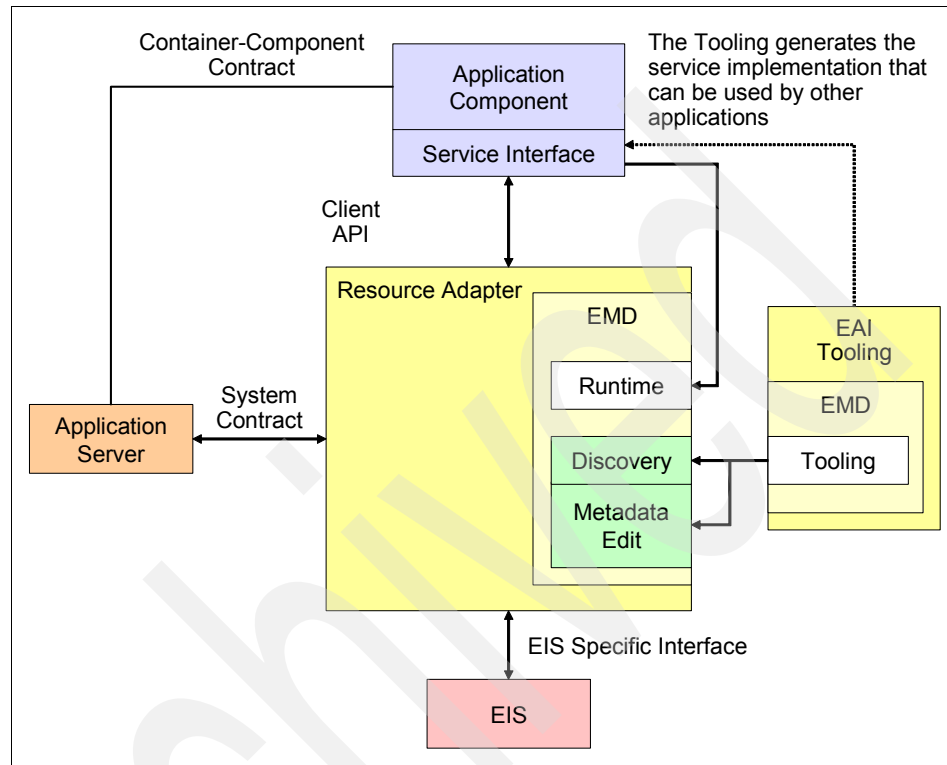


Figure 6-1 Enterprise Service Discovery architecture

6.3.1 J2CA components related to EMD

The various J2CA components related to EMD are:

- Application server

It is the J2EE application server that supports the standard resource adapter based on J2EE CA specification 1.5. All the application and resource adapters work inside the application server. In this document, we use IBM WebSphere Process Server V6.

- Application component

This is J2EE application that works on application server. Most of the integration business logic is implemented in the application component. It uses Common Client API to call the EIS services, which provided or wrapped by the resource adapter.

- ▶ Resource adapter

A JCA resource adapter is a system-level software driver for connecting to a specific EIS.

- ▶ EIS

EIS is an existing enterprise system, which contains various data and provides services.

- ▶ EAI Tooling

EAI Tooling is a tool for integration developer. In this document, we use IBM WebSphere Integration Developer V6.

- ▶ Service interface

Service Interface describes services provided by the EIS through the adapter. The service interface specifies the operations and parameters for the service. In order to work within WebSphere Business Process, the service interface must also be SCA compliant. SCA compliant interface allows other service components in WPS to invoke adapter operations.

- ▶ EMD

EMD is a component of resource adapter that implements the adapter side of the EMD specification. It contains business logic to connect to EIS, generate service interfaces and create business objects from data in the EIS.

- ▶ EMD - runtime

This interface extends the CCI interface for getting connection to EIS. The reason we need this component is that the EMD also need to connect to EIS to get the metadata information. Usually it reuses some adapter outbound implementation to get an EIS connection.

- ▶ EMD - discovery

This part includes a series of interfaces between EAI Tooling and resource adapter. Using these interfaces, the Tooling can manage the connection, browse the metadata in EIS, and create/edit the service description, which represents the services and data structures in EIS.

- ▶ EMD - Metadata edit

This interface allows the EAI Tooling to edit the configurations for the service description.

- ▶ EMD - Tooling

The EAI tools implements the EMD tooling contracts specified in the EMD specification. The EMD Tooling provides User Interface as well as the generating process control for the EMD. It also provides some utility APIs for the resource adapter EMD, which can perform progress report, logging, tracing.

6.3.2 Relationships between components

The various relationships between components are:

- ▶ Application server and application
They use J2EE container-component contract to communicate. Application server provides the container for the applications.
- ▶ Application server and resource adapter
They use System Contract of J2CA 1.5 specification to communicate. The application server provides the container for resource adapters.
- ▶ Applications and resource adapter
They use Common Client API of J2CA 1.5 specification to communicate.
- ▶ Resource adapter and EIS
Resource Adapter uses the specific APIs of the corresponding EIS to interaction with EIS.
- ▶ EMD Tooling and adapter EMD discovery
Adapter EMD Discovery implements the methods that are used to connect to EIS and generate services and business objects while EMD Tooling provides the user interface. The EMD tooling is the front end of ESD while the adapter EMD component is the back end.
- ▶ EMD Tooling and EMD Metadata Edit
EMD Metadata Edit implements the methods, which are called by EMD Tooling to edit the configuration for service description and business objects.
- ▶ EAI Tooling and service interface
The EAI Tooling uses EMD Tooling to generate the service interface. Then EAI Tooling can implement the service interface that can be used by applications. Note that during generating service interface, EAI Tooling does not communicate with applications. Instead, it communicates with resource adapter EMD to collect the information from EIS.

6.3.3 J2EE Roles of Enterprise Metadata Discovery

Figure 6-2 describes the important J2EE roles of EMD. For more information about the roles, refer to the EMD specification white paper, available here:

<http://www-128.ibm.com/developerworks/library/specification/j-emd/>

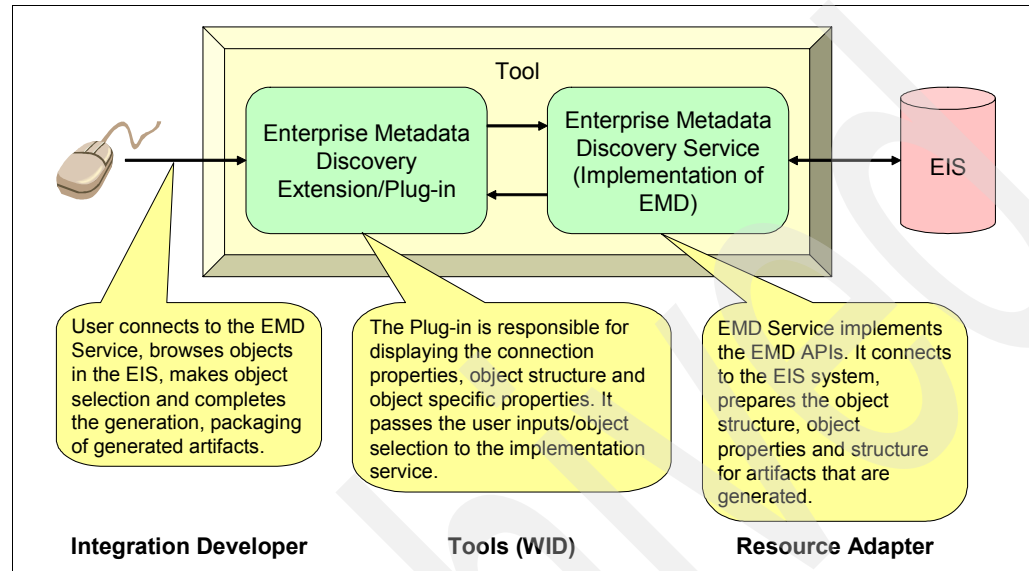


Figure 6-2 Enterprise Service Discovery roles

Details of the roles shown in Figure 6-2 are:

- **Tools developer**

Tools developers are responsible for implementing the Tools interfaces in EMD definitions. Tools provide the User Interface and some basic functions for EMD. In IBM WebSphere Integration Developer V6, such tools have been implemented.

- **Resource adapter developer**

The resource adapter developers are responsible to implement the adapter side of the EMD contract for runtime, discovery and metadata edit.

- **Integration developer**

Integration developers create integration solutions. When they develop a solutions that involve an adapter and the adapter supports EMD, they can use the ESD tools provided by the EAI tooling. For example, they use ESD in WID. From the tool they can browse the data/metadata in the EIS, select all or part of the node for business object generation and let the tool to auto generate service artifacts.

6.4 Process of generating service by using ESD

This part describes how to perform a service description by using ESD. Usually this is typically performed by the integration developer. The process begins when the integration developer launches the ESD Tool. Figure 6-3 shows the main steps in the process of creating a service.

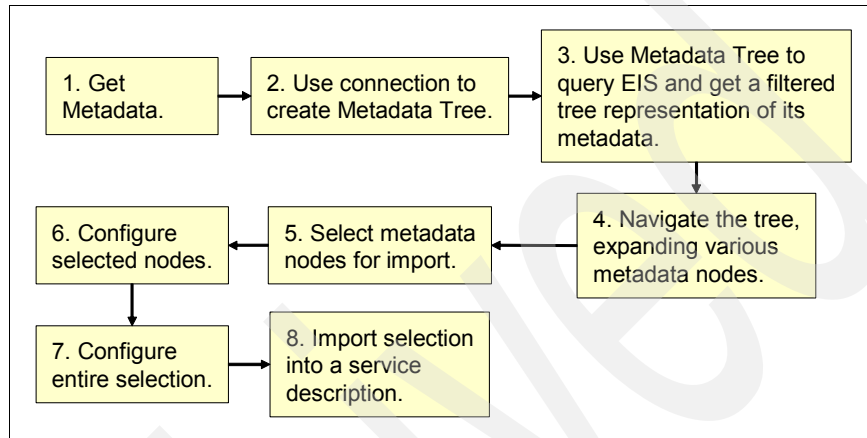


Figure 6-3 Using Enterprise Service Discovery to create a service

The steps are as follows:

1. Tools show the required connection configurations to the user. The user fills in these fields with the correct EIS information.
2. After getting the required information, the Tools connect to EIS and create a metadata tree by calling the interfaces that the resource adapter provides. The metadata tree is a tree model to represent the data structures in the EIS.
3. ESD users can set filters and run query on the metadata tree. Then, they can get a specific tree with the required data.
4. ESD users can navigate the tree with multi level nodes structure.
5. ESD users can select metadata nodes for import.
6. When adding a node, a dialog box might open to let users input the specific configuration for that node.
7. After selecting all the required notes, users need to provide configurations for the whole import, such as the path of the output files, the namespace and so on.

8. In this step, EMD users need to provide information for the service description other than the data structure in EIS. It could be a related project in Tools, runtime connection properties for destination applications and so on. After finishing all the above preparations, Tools generate the service interface with all the data structures it needs.

After finishing the above steps, Integration Developers can package and deploy the application with the service description.

6.5 Process of developing EMD for a resource adapter

In this section, we describe the steps in developing the EMD component for an adapter. Implementation of EMD is described in Chapter 13, “Implementing Enterprise Metadata Discovery” on page 393. In that chapter, we provide a sample implementation of adapter EMD component.

Instead of starting EMD development from scratch, we can leverage the default implementation of EMD provided by the Adapter Foundation Classes. When using Adapter Foundation Classes to develop EMD, we only need to extend the base EMD implementation classes to perform our EIS specific tasks. For more information of Adapter Foundation Classes see Chapter 10, “Adapter design and specification” on page 237.

For implementing EMD for a resource adapter, basically we must do the following:

1. Set up your development environment as described in Chapter 8, “Setting up the development environment” on page 159.
2. Design what metadata in EIS needs to be shown to the user and generated by EMD. This is the design-level job. Usually, EMD tries to generate the same service interfaces that adapter runtime (outbound and inbound) needs. So, before implementing EMD, it is required that you have a clear understanding which services in EIS need to be exposed to use.
3. Use WAT to generate EMD code stubs for your resource adapter.

Note: It is recommended that you implement EMD after you have implemented adapter outbound processing. This is because EMD must reuse some of the classes implemented in outbound for EIS connection. Otherwise, you need to write EIS connection code during EMD development.

4. Create the deployment descriptor for EMD. The deployment descriptor is an xml configuration file for EMD. WAT 6001 does not create this file automatically during EMD code stub generation. It needs to be manually created.
5. Create the application-specific information schema file. The purpose of this schema file is to constrain the metadata annotations in the business object definition file.
6. Become familiar with the utility APIs provided by EMD Tooling. The EMD Tooling in WID provides many utility APIs including log, trace, monitor and and so on, which can be used by resource adapter EMD.
7. Implement the EMD stubs classes generated by WAT. Here, almost all the methods that need to be implemented have generated by WAT. We need to understand the desired behavior for each method in the stub classes and provide our EIS specific implementations.
8. Unit test the EMD implementation on EAI tool.

Exceptions, logging, and tracing

When developing software, we often encounter errors and exceptions. Developing a custom adapter is no different. To develop an adapter, we have to work with sophisticated software tools and test the adapter in a complex environment. To assist us with problem determination and problem solving when developing resource adapters, this chapter discusses adapter logging and tracing.

This chapter discusses the following topics:

- ▶ Adapter exception handling
- ▶ Adapter logging and tracing
- ▶ Business events

7.1 Adapter exception handling

In software development, the term exception refers to a programming construct that is designed to handle conditions which can change the normal flow of program execution. Exception objects and exception traces can convey important information about why the exception has occurred and where in the code it occurred. During adapter development, we should follow these guide lines.

7.1.1 Exception creation during adapter development

The root interface defined for the connector architecture exception hierarchy is *ResourceException*. When we write exception classes that have a specific action or category, the exception classes should always subclass *ResourceException*. For example, a *ConnectionFailedException* might represent an adapter-specific connection failure, so when we write *ConnectionFailedException* class for exceptions in the connection category, we need to extend *ResourceException* class. This section shows some examples of Service Provider Interface exceptions that extends *ResourceException*. They are defined in the JCA specification:

- ▶ `javax.resource.spi.EISSystemException`
- ▶ `javax.resource.spi.IllegalStateException`
- ▶ `javax.resource.spi.ResourceAdapterInternalException`
- ▶ `javax.resource.spi.LocalTransactionException`

In addition to creating classes for each exception category as described previously, we should add properties to exceptions that can be used for secondary programmatic debugging. For example, we should add the following properties to *ConnectionFailedException* as well as providing *getter* and *setter* methods:

- ▶ `hostName`
- ▶ `portNumber`
- ▶ `userName`
- ▶ `password`

Having these properties in the exception class allows other components outside of the adapter to process the exception. For example, these components can invoke the exception instance's *getHostName()* method to retrieve the host name of the EIS or invoke the *getPortNumber()* method to retrieve the port number. With this information the component could restart the EIS if it is down.

The *toString()* and *getMessage()* methods of our new exception classes return a meaningful string if parameters have to be filled with the message, for example:

"Connection Failed for <hostname> on <port number> for <username>".

7.1.2 Exception generation at runtime

When creating or constructing a new exception, it is recommended that you:

- ▶ Include a suitable message with the new exception. This propagates up to the application server (and presumably the user console). For example:

```
throw new java.lang.IllegalStateException("Message endpoint factory  
specified is not currently activate for this adapter.");
```

- ▶ Print the stack trace of the exception using the trace API. This allows support team to see details related to the exception.
- ▶ When appropriate, the exception messages raised should have a corresponding high-level message in the LogMessages.properties file. The developer should log this high level message after raising the exception so that users can see an explanation of the error in their native language.

```
logUtils.log(Level.SEVERE, LogUtilConstants.BASE_RBUNDLE, CLASSNAME,  
method, "0010", engineData);
```

In the following section we provide more explanation on logging and tracing.

7.2 Logging and tracing

This section explain logging and tracing when developing a resource adapter.

7.2.1 Logging

To help with adapter runtime problem determination and problem resolution, the adapter code need to write log messages to a log file. The log file captures information such as adapter start and stop, success or failure, configuration errors, and bugs in the adapter. The log file is intended for adapter end users, system administrators, customer service engineers, and adapter development teams.

JCA 1.5 specification logging and tracing is not robust, it provides resource adapters with only a `PrintWriter` for the purpose of writing all messages. This limited approach lends itself to ad hoc standards being adopted by different adapters, fails to provide proper encapsulation of messages for different target users (customers versus administrators for example) or enable support for monitoring tooling.

The Adapter Foundation Classes provide a set of classes to provide more robust and consistent functionality across different deployment scenarios. The logging utility classes provide a proxy class on top of the Java Logging API (JSR47). This hides the specifics of the JSR 47 APIs and provides a more compact and consistent API for developers to use. This also helps ensure consistency in the

information logged and traced in all the resource adapters. It is highly recommended that you use Adapter Foundation Classes when developing your custom adapter. The logging classes provided for Adapter Foundation Classes are located in the package:

`com.ibm.j2ca.extension.logging`

See Adapter Foundation Classes' Javadoc™ for more information about this package.

When writing to the log file in our adapter code, we can specify the severity of the log entry, see example below:

```
logUtils.log(Level.SEVERE, LogUtilConstants.BASE_RBUNDLE, CLASSNAME, method,
"0010", engineData);
```

Adapters can be quite complex and can produce many entries in the log file. By setting the logging levels, we can specify the level of logging we want at runtime. For example during development and unit testing we might want to see detail logs, while in a production environment we might want to see severe and fatal log entries only to reduce the number of entries in the log file. Table 7-1 shows the available logging levels.

Table 7-1 Logging levels

Level	Indicator	Content/Significance
Fatal	F	Task cannot continue. Component cannot function
Severe	E	Task cannot continue. Component can still function. This also includes conditions that indicate an impending fatal error - i.e. reporting on situations that strongly suggest that resources are on the verge of being depleted.
Warning	W	Potential error or impending error. This also includes conditions that indicate a progressive failure - for example, the potential leaking of resources.
Audit	A	Significant event affecting server state or resources
Info	I	General information outlining overall task progress.
Config	C	Configuration change or status
Detail	D	General information detailing subtask progress

EMD logging

EMD classes would use the same class provided by base classes for JCA part of adapter. It would provide a separate mechanism for initialization as the logger

initiation for EMD is performed by the EMD tool and does not need initiation in adapter. The methods that should be used for logging would be the same as used for JCA part:

- The resource bundle name for base classes would be:

```
com.ibm.j2ca.extension.emd.LogMessages.properties
```

- Individual EMD implementations would have it as:

```
com.ibm.j2ca.<adapterName>.emd.LogMessages.properties
```

7.2.2 Tracing

Trace messages should be used to convey information that is intended for support teams and for adapter developers. Tracing can help us follow the flow of adapter code execution and quickly determine the point of failure in the code. Tracing messages do not need to be translated.

There are three trace levels, as Table 7-2 shows. This allows users to adjust the level of tracing granularity. The following guidelines should be consulted to determine which trace level to assign to a given trace message.

Table 7-2 Trace Message Levels

Level	Indicator	Content/Significance
Fine	1	General trace. Includes broad actions being taken by the adapter such as establishing a connection to the EIS, converting a event in the EIS to a business object (only key values), processing a business object (only key values).
Finer	2	Detailed trace that provides more granuled information about the logic being performed by the adapter, including the various API calls being made to the EIS and any parameters or return values.
Finest	3	This is the most detailed level and should include method entry / exit / return values. Complete business object dumps should be included. At this level, all detail needed to debug problems should be provided.

Writing a trace message

The `trace` method of the `LogUtils` class should be used to generate a trace message.

LogUtils class provides a number of trace methods:

- ▶ `public void trace(java.util.logging.Level level, java.lang.String classname, java.lang.String method, java.lang.String msg)`
- ▶ `public void trace(java.util.logging.Level level, java.lang.String classname, java.lang.String method, java.lang.String msg, java.lang.Exception e)`
- ▶ `public void traceDataObject(commonj.sdo.DataObject busObj, java.lang.String classname, java.lang.String method)`
- ▶ `public void traceMethodEntrance(java.lang.String classname, java.lang.String methodSignature)`
- ▶ `public void traceMethodExit(java.lang.String classname, java.lang.String methodSignature)`

Usage example

Here is one example of a trace method:

```
logUtils.traceMethodEntrance(CLASSNAME, METHOD);  
logUtils.traceMethodExit(CLASSNAME, METHOD);
```

7.2.3 Message files

An adapter message file is a Java resource bundle file. This file is used to externalized messages in the adapter code. In this file, there is a key for every message. So instead of harding the message in the source code, we reference the key instead. We can have the message file translated to other languages. At runtime, Java resolves the resource key to the language of the current operating system the adapter is running on. See examples below:

1. `void log(java.util.logging.Level, int bundleType, String classname, String method, String msgKey)`
2. `void log(java.util.logging.Level, int bundleType, String classname, String method, String msgKey, Object[] params)`

Creating the message file

The message file must be placed in this location:

```
<adapter package>/LogMessages.properties
```

Any translated message file must be place in the same directory and follow the naming convention:

```
<adapter package>/LogMessages_xx.properties
```

Where xx represent the locale of the file. For example, a translated French file is named:

<adapter package>/LogMessages_fr.properties

All the messages must contain the following three parts:

► Message Id

The message id must follow the following format NNNNNmmmmS, where NNNNN is a five letter component prefix, mmmm is the message number and S is a type identifier to identify the type of the message. That is, in CWxxx0000E, the component prefix is CWxxx, the message number is 0000 and E identifies the message as an error message. For the component prefix, we use CWx to identify the family of adapters and adapter-related components and two characters to identify individual adapters. The two characters used for base classes are 'BS'. Adapter developers should use the internal document in place to assign the two character IDs for the adapters.

► Explanation

Provide a more in-depth explanation of the message with the assumption that the user can be unfamiliar with the entire meaning of the base message itself. The explanation is not to be used as a crutch for a poorly written base message text, but instead is intended to be viewed as the first level of help documentation for users.

► User Action

For any given explanation of what went wrong, there are usually one or more actions that the user can take to rectify the situation, or to ensure that it does not happen again. The User Action field should provide fairly detailed information about what the user should do. Again, think of this as the first level of help documentation for users.

Example resource bundle file entry

The Log Message file follows the WebSphere Application Server convention. It has, for each message, a <key>=value pair, user action and an explanation. In Example 7-1, {0} mentions some arguments to be passed dynamically.

Example 7-1 Sample of Log Message file

```
0001=RM0001I: Starting adapter.  
0001.useraction=No action required.  
0001.explanation=Starting adapter.
```

```
0004=RM0004W: Failed to shutdown event manager: {0}  
0004.useraction=No action required.  
0004.explanation=Failed to shutdown event manager.
```

7.2.4 Business events

Another type of messaging, business events, are intended for consumption by monitoring tooling. These events are based on the Common Business Event (CBE) model and should not be confused with inbound business events that the adapter generates to notify downstream processes of changes in the back-end EIS.

CBE events convey information about the adapter state. This can include IT-level information such as whether or not the adapter is running, as well as business-level information such as key information about the requests an adapter is processing. CBE events are published to the Common Event Infrastructure (CEI) where they are persisted and consumed by tooling.

Adapter developers do not generally need to concern themselves with generating such events unless there is a clear business requirement. The adapter foundation classes automatically generate many of the events requested by customers.

Writing a CBE event

To generate a CBE event, you must first create and populate an instance of *CBEEngineData*. This is then passed along with a log message via the two signatures of *LogUtils.log* that accept CBE engine data. The information contained in the log message along with the additional CBE data is published to the CEI bus.

Defining custom events

If the existing events generated by WebSphere adapters are insufficient for your adapter needs, you can define additional adapter events.

1. Create the event definitions file for the custom events. Define the event sources. An event source is the particular functional area of the adapter that is monitored. Extend class *J2CAEventSource* to define new event sources.
2. Define the event types. An event type is composed of an event source and a particular state that is valid for that event source. Extend class *J2CAEventType* to define new event types implementing method *shouldMonitor* and *getPayloadTypes*.
3. Define the payload type elements. A payload event type is the data associated with the event type. Extend class *J2CAPayloadElementType* to create the payload element types. A critical element of this implementation is providing a *J2CAPayloadElementTypeSerializer* implementation that can convert the payload element to a representation that fits the CBE model.

4. Publish the event definitions to the CEI catalog. Refer to the Web site for more details:

<http://publib.boulder.ibm.com/infocenter/wbihelp/index.jsp?topic=/com.ibm.wasee.doc/info/ee/wei/ref/rea0140.html>

Archived



Part 2

Custom adapter development

ARCHIVED

Setting up the development environment

In this chapter we take you through the process of installing the software required for your adapter development environment on the Microsoft Windows platform.

In this chapter we explain:

- ▶ Installing WebSphere Integration Developer V6.0.1
- ▶ WebSphere Adapter Toolkit V6.0.0.1 overview
- ▶ Installing WebSphere Adapter Toolkit V6.0.0.1
- ▶ Creating a Hello World adapter using WebSphere Adapter Toolkit V6.0.0.1
- ▶ Twineball sample adapter

Important: In order to successfully install the required software for your adapter development environment, you must install the software in the order described in this chapter.

IBM software can be installed interactively using a setup wizard or installed silently from the command line with a preconfigured silent install response file. Only interactive install using the setup wizard is described in this book. Refer to the installation documentation that comes with IBM software for complete installation instructions.

8.1 Installing WebSphere Integration Developer V6.0

To develop custom adapters, we use WebSphere Integration Developer and WebSphere Adapter Toolkit as our development environment.

WebSphere Integration Developer setup wizard guides you through the installation process through a series of install dialogs. Follow these steps to install WebSphere Integration Developer V6.0.1:

1. Launch the setup launchpad.

To begin installation, navigate to the directory of the installation media and double-click the launchpad file **launchpad.exe**. The setup launchpad opens as shown in Figure 8-1. Click **Install IBM WebSphere Integration Developer V6.0.1** on the launchpad to start the setup wizard.

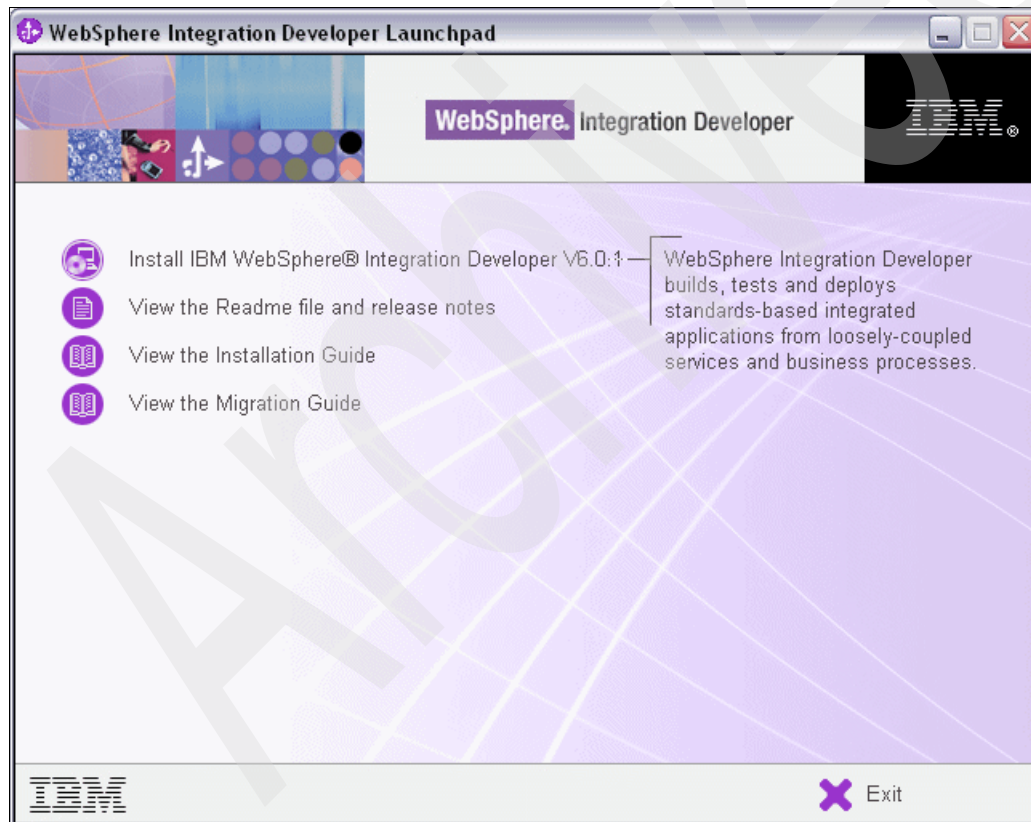


Figure 8-1 Setup launchpad

2. The welcome dialog box of the setup wizard opens, as shown in Figure 8-2. Click **Next** to continue.

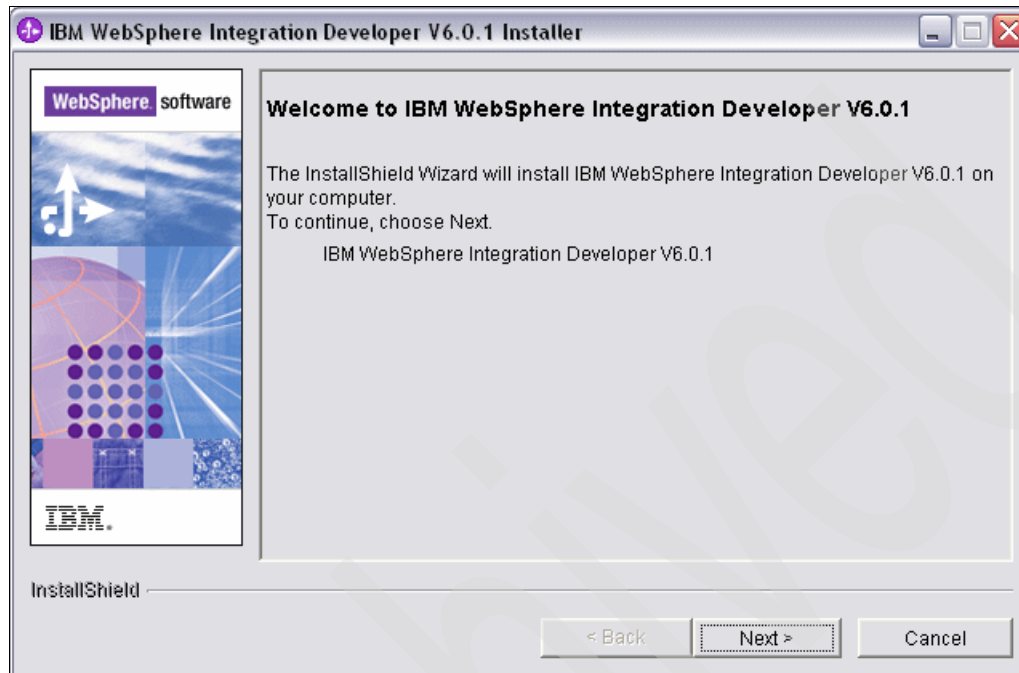


Figure 8-2 Welcome dialog

3. The license agreement dialog box appears as shown in Figure 8-3. Click **I accept the license terms in the license agreement** to accept the license agreement.

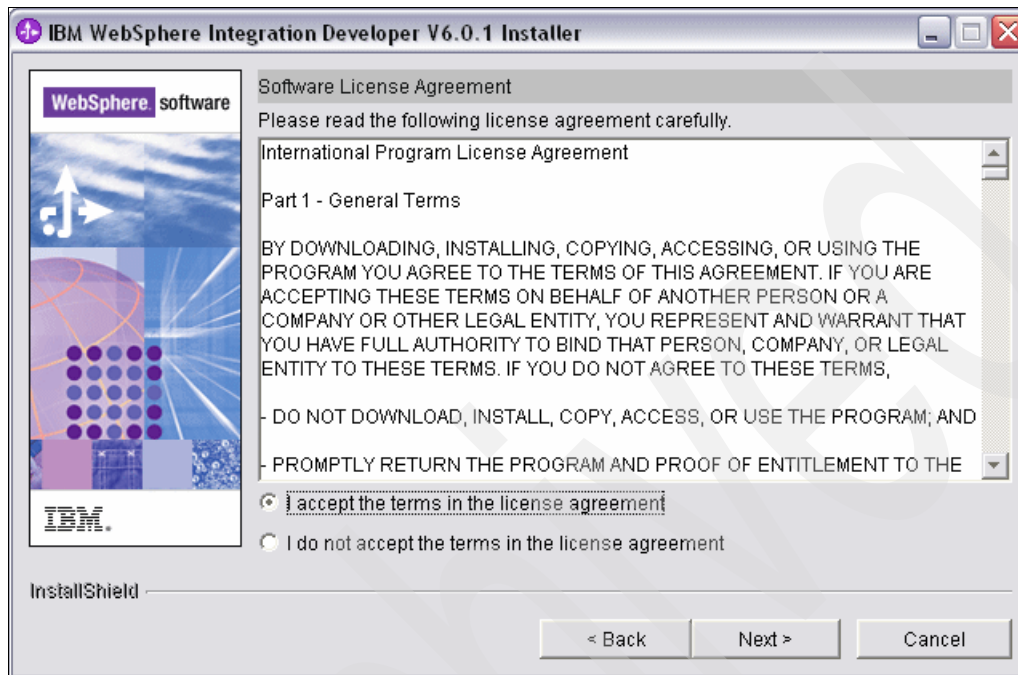


Figure 8-3 License dialog

4. Click **Next** to continue.

5. The install location directory dialog box opens as shown in Figure 8-4. Click **Next** to accept the default install location directory. The default install location is C:\Program Files\IBM\WebSphere\ID\6.0.

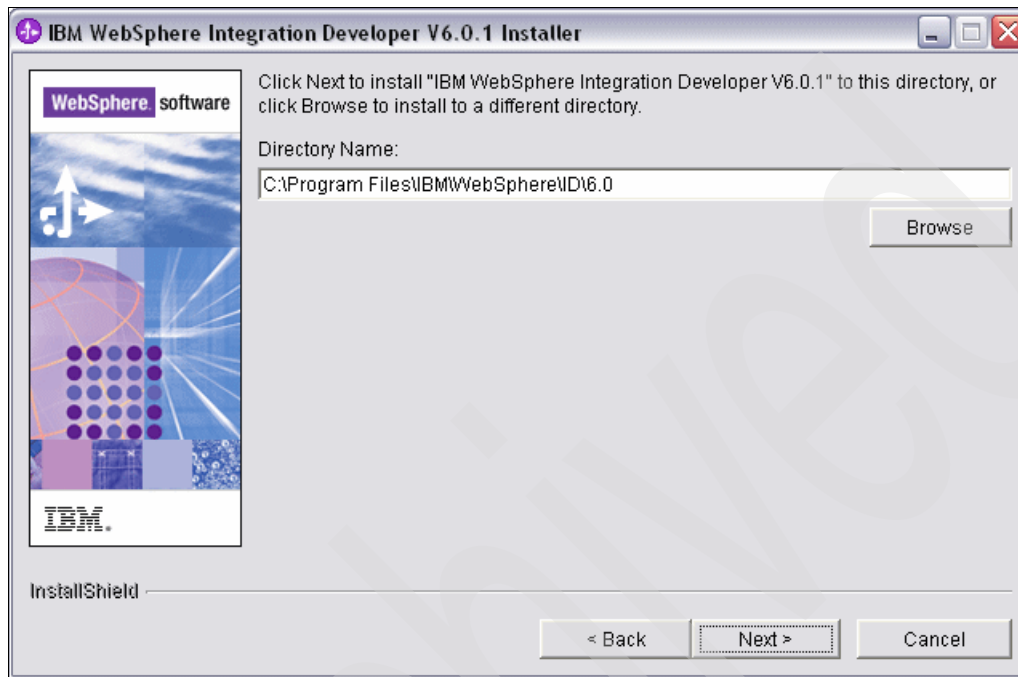


Figure 8-4 Installation location directory dialog

6. The first product feature selection dialog box opens as show in Figure 8-5. Select **IBM WebSphere Process Server V6.0.1 Integrated Test Environment**.

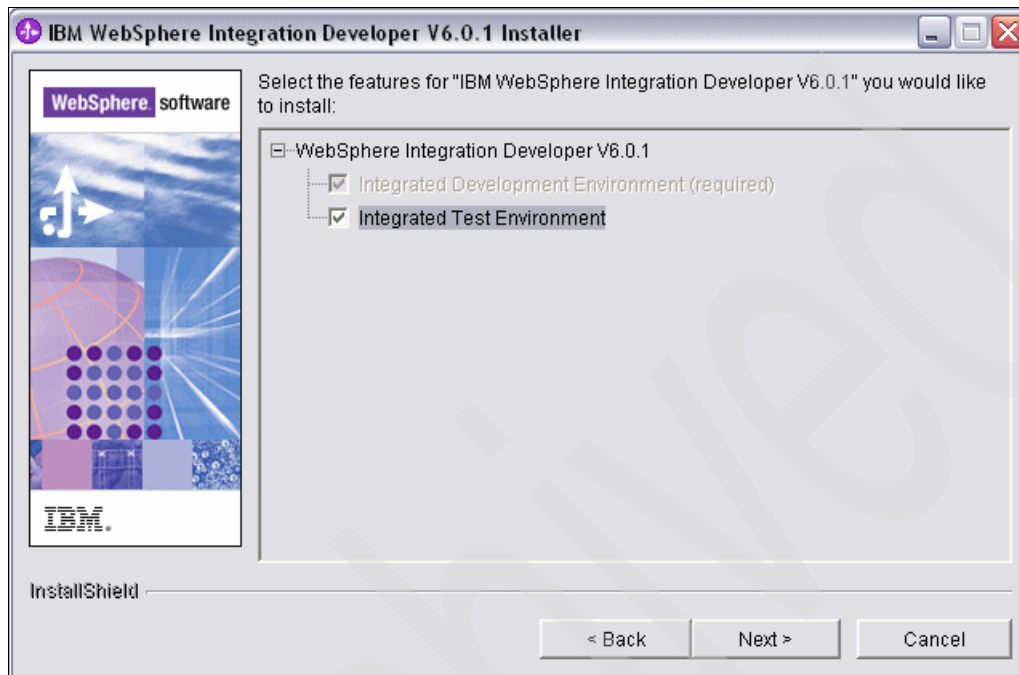


Figure 8-5 Feature selection dialog 1

7. Click **Next**.

8. The second product feature selection dialog box opens, as shown in Figure 8-6. Ensure **WebSphere Process Server** is selected. Click **Next**.

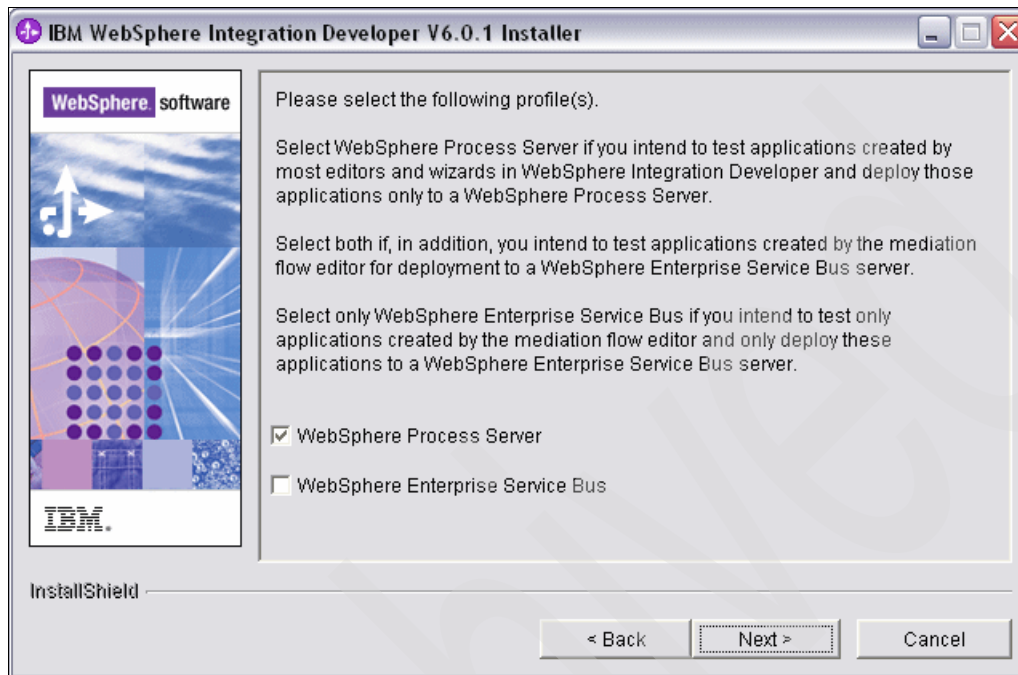


Figure 8-6 Feature selection dialog 2

9. The preinstallation summary dialog box opens as shown in Figure 8-7. Review the information on this pane, then click **Next** to continue.

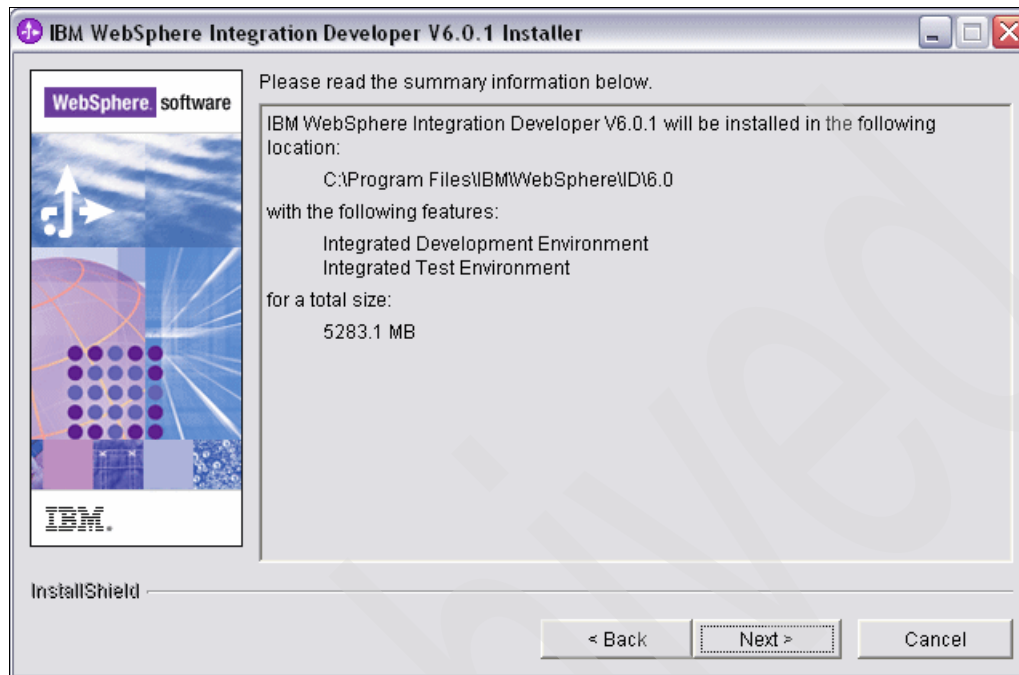


Figure 8-7 Pre-install summary dialog

10. The installation dialog box opens as shown in Figure 8-8. This pane shows the installation in progress. When installation is complete, the setup wizard automatically opens the next dialog box.

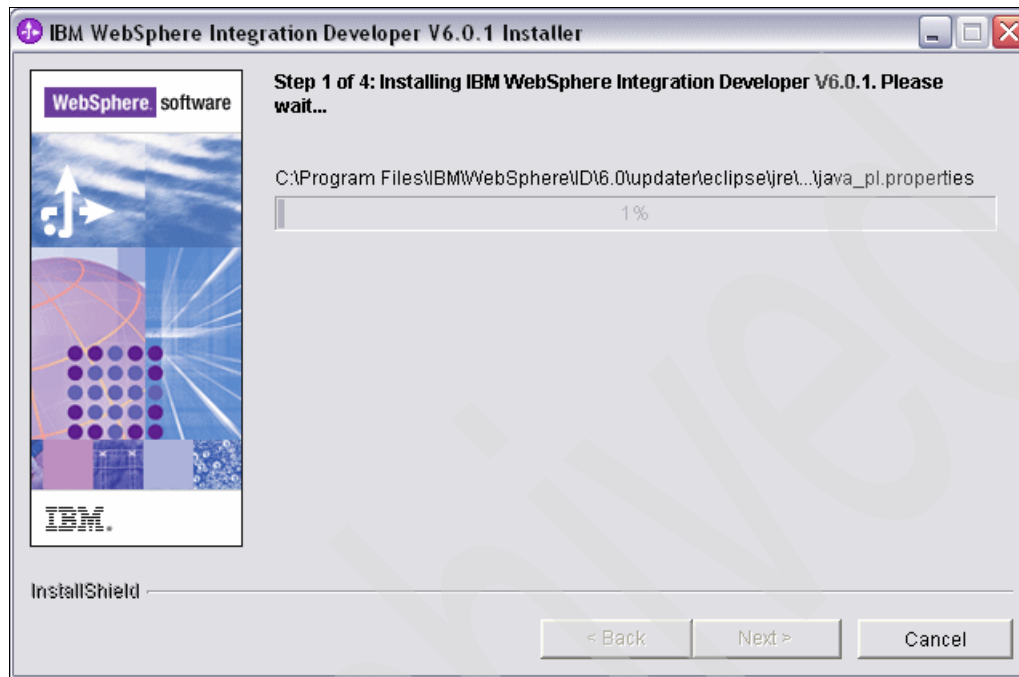


Figure 8-8 Installation dialog

11. The post installation summary dialog box opens as shown in Figure 8-9.
Review the information then click **Next** to continue.

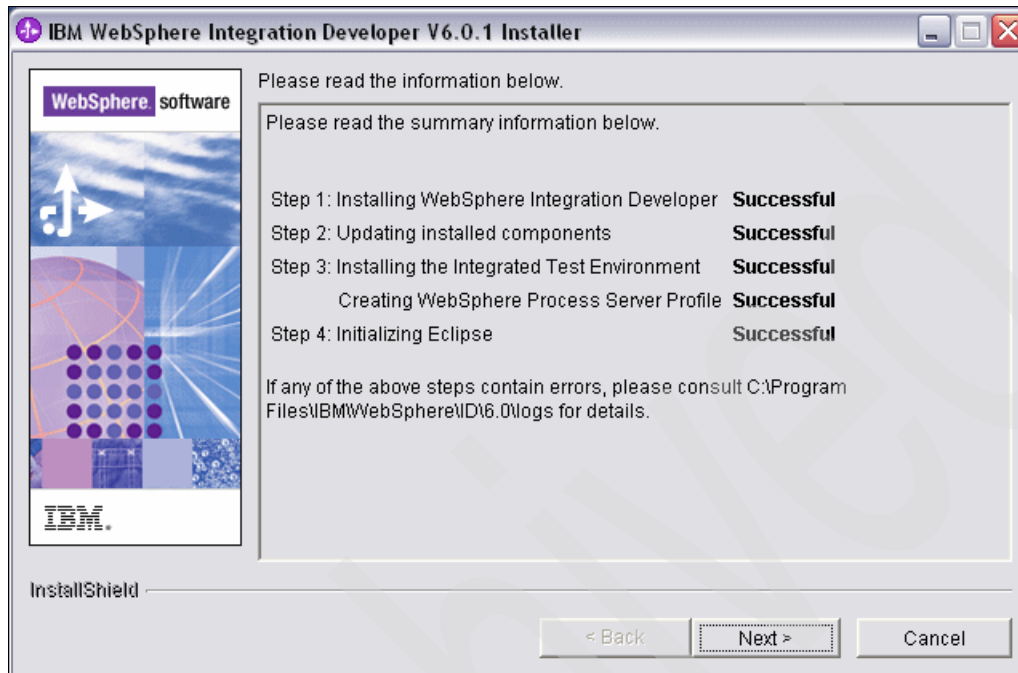


Figure 8-9 Post install summary dialog

12. The readme pane opens, as shown in Figure 8-10. Click **Next** to review the readme file in a browser.

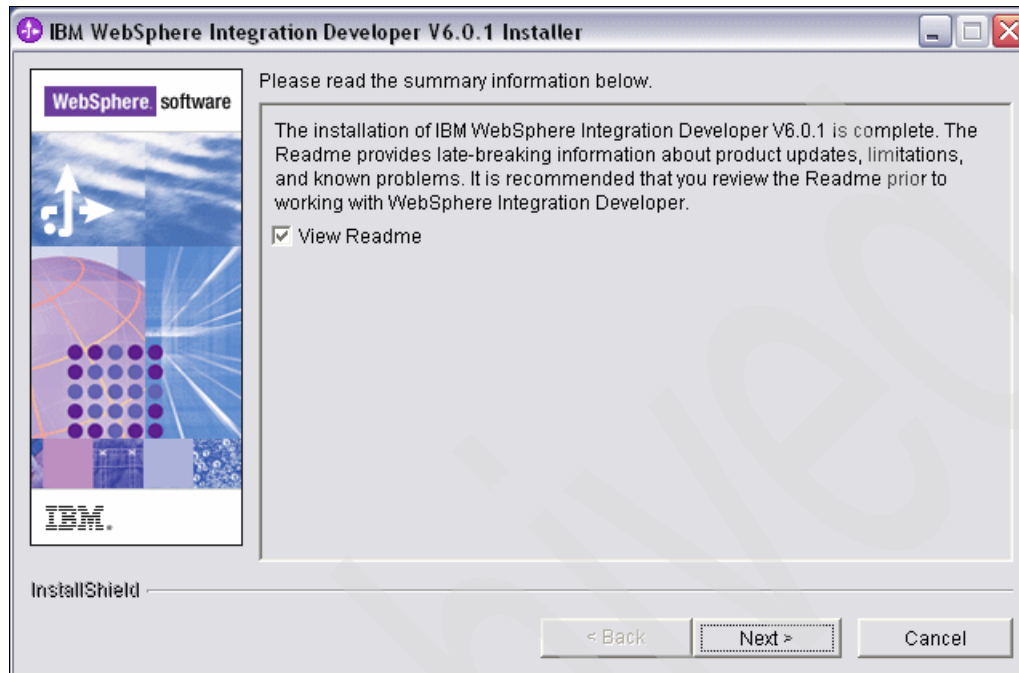


Figure 8-10 Readme dialog

13. Close the browser after reading the file.

14. Click **Next** on the installation summary dialog box as shown in Figure 8-11 to go to the next panel.

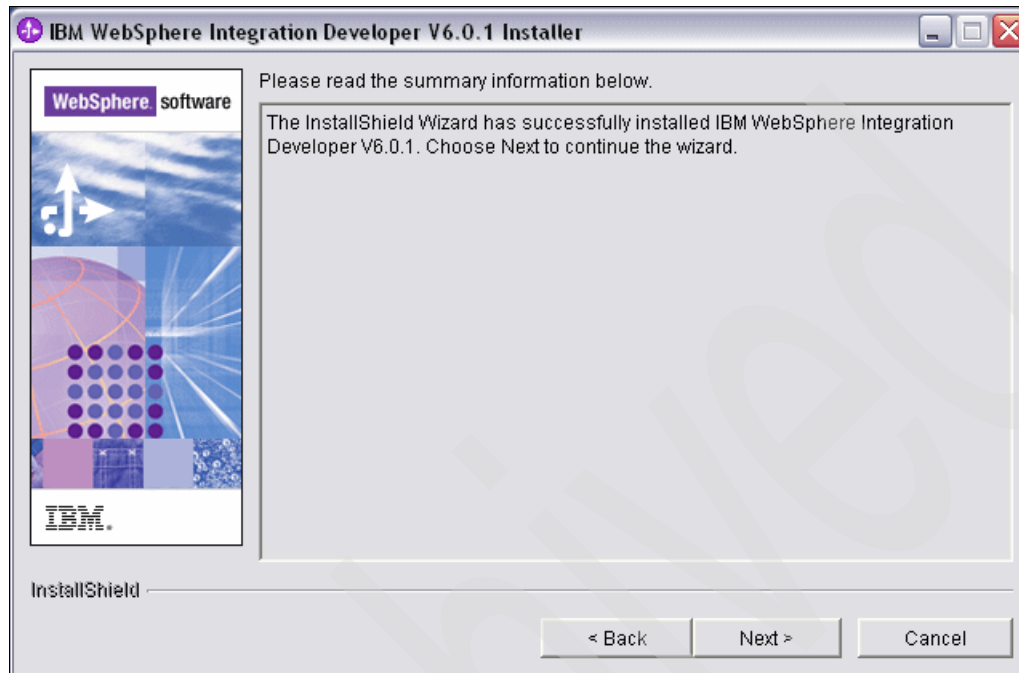


Figure 8-11 Finish dialog

15. Click **Finish** on the last installation dialog box as shown in Figure 8-12 to launch Rational Product Updater.

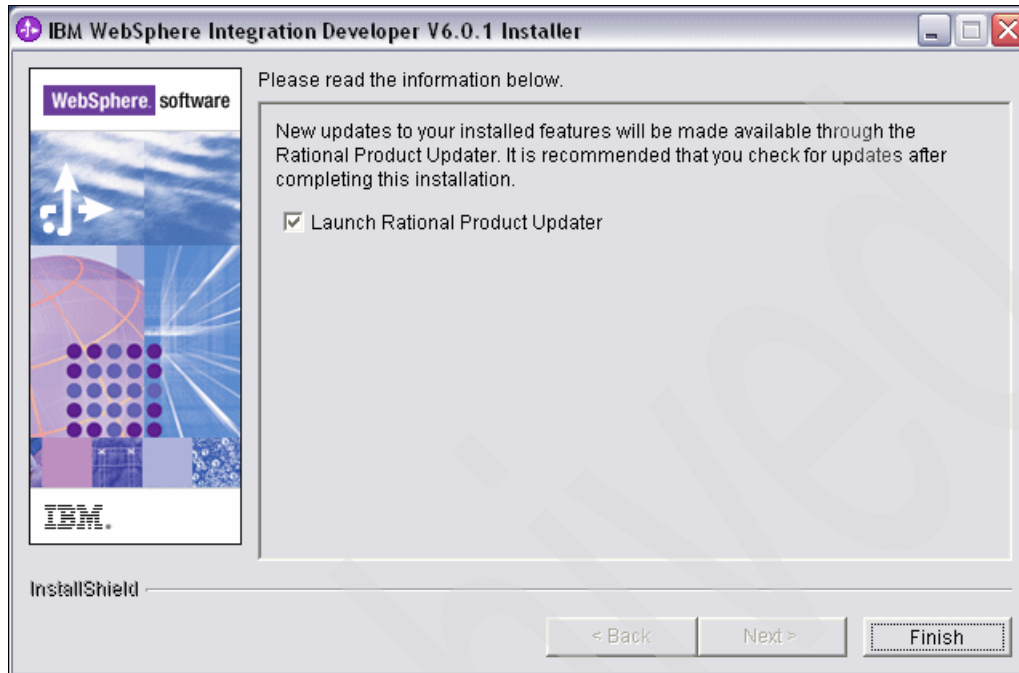


Figure 8-12 Launch Rational Product Updater from installer

16. Figure 8-13 shows the Rational Product Update. We can use the updater to update WebSphere Integration Developer with the latest fixes. Click **Update** tab to view available updates.

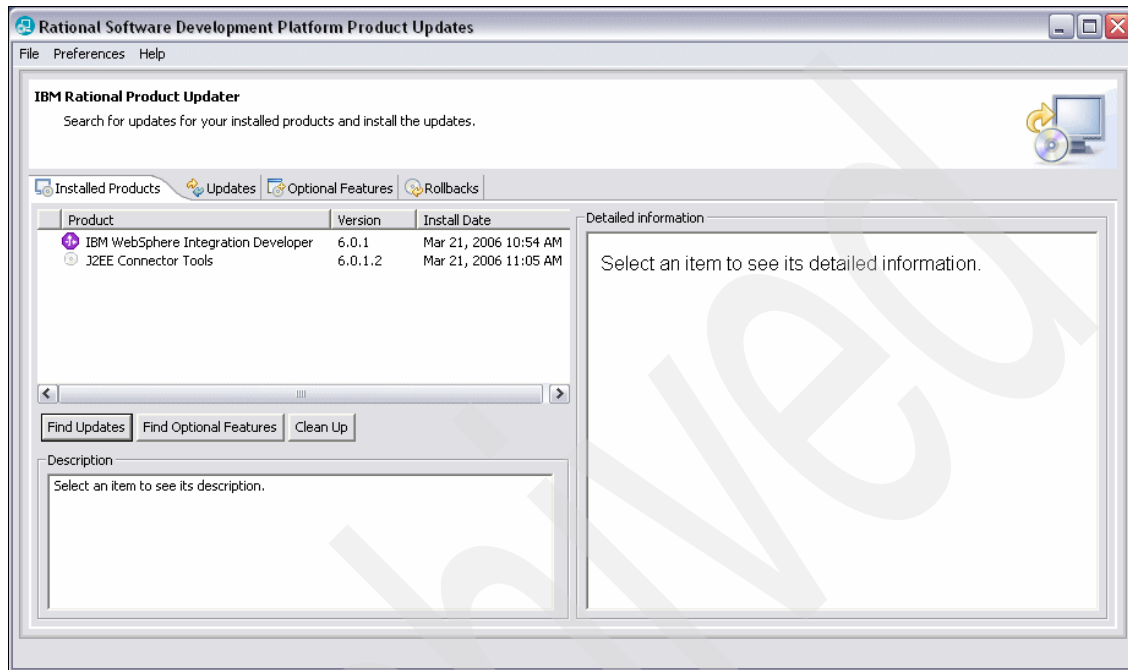


Figure 8-13 Rational Product Updater

17. Click **Install Updates** as shown in Figure 8-14 to start the Update process.

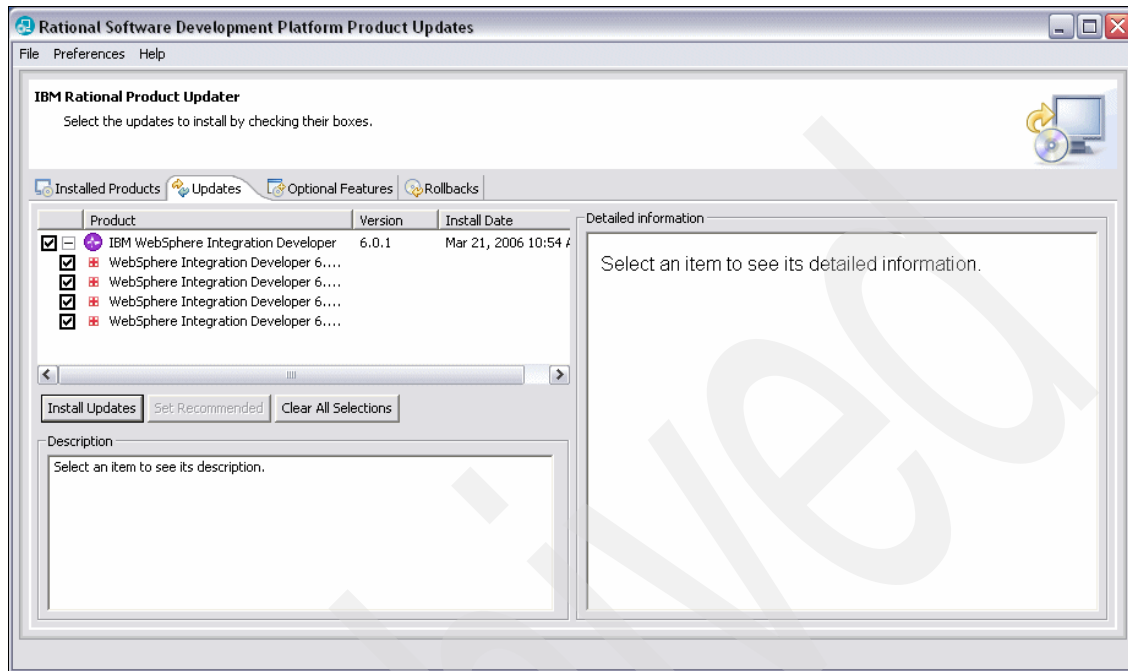


Figure 8-14 Available updates

18. Accept the license agreement as shown in Figure 8-15, then click **OK** to start downloading and installing the updates. Depending on the amount of updates, this process might take some time.

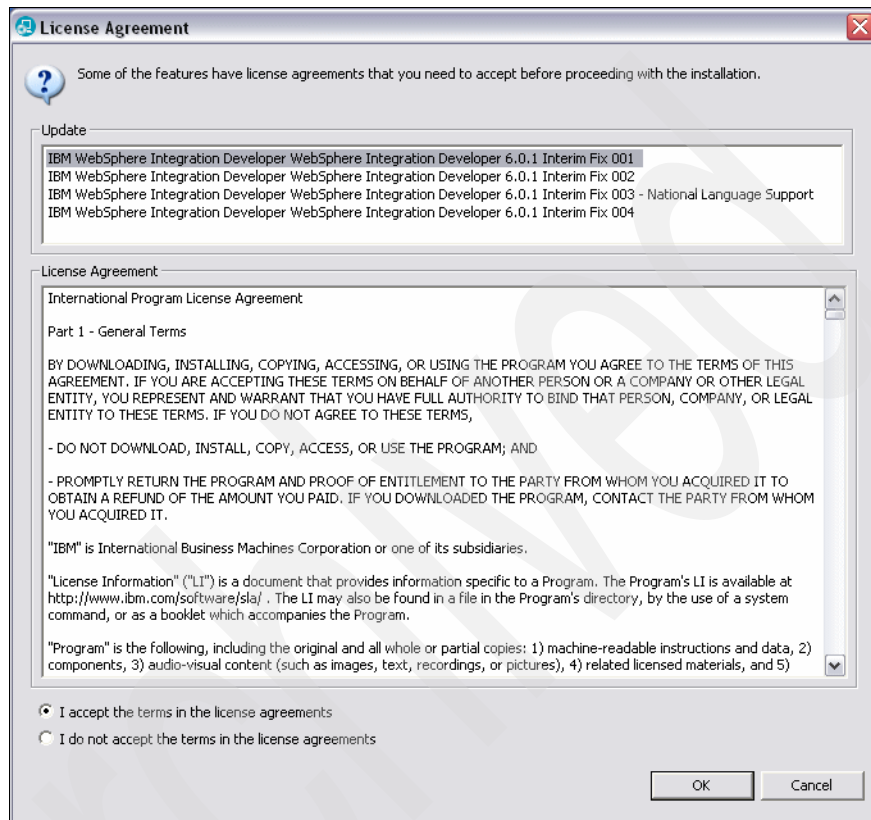


Figure 8-15 License agreement dialog for updates

8.2 WebSphere Adapter Toolkit overview

The IBM WebSphere Adapter Toolkit contains the tools you need to create a resource adapter. The toolkit itself is an eclipse plug-in deployed to WebSphere Integration Developer. It runs in the context of WebSphere Integration Developer's integrated development environment.

The toolkit has the following features:

- ▶ An adapter development wizard to guide you to create an adapter development project. It can create the following type of adapter projects:
 - IBM WebSphere Resource Adapter.
Create this type of adapter project if you want your adapter to run on WebSphere Process Server.
 - J2EE J2CA Resource Adapter.
Create this type of adapter project if you want your adapter to run on any JCA compliant application server other than WebSphere Process Server.
- ▶ Generate Java code stubs that the adapter developer must implement to create a working adapter.
- ▶ Provide an adapter deployment descriptor editor.
- ▶ Provide WebSphere Adapter Foundation Classes and put the foundation classes jar file in the adapter project's build path when a new WebSphere Resource Adapter type adapter project is created.
- ▶ A working sample adapter call TwineBall to help you develop your own resource adapter.

For more details on the toolkit, see the *IBM WebSphere Adapter Toolkit V6.0.0.1 User Guide*:

<http://www-128.ibm.com/developerworks/websphere/downloads/wat/>

8.3 Installing WebSphere Adapter Toolkit V6.0.0.1

WebSphere Adapter Toolkit V6.0.0.1 is an eclipse plug-in. The toolkit installer deploys the plug-in to WebSphere Integration Developer. We use WebSphere Integration Developer to develop custom resource adapters. You can download WebSphere Adapter Toolkit for free from:

<http://www.ibm.com/developerworks/websphere/download/wat/>

WebSphere Adapter Toolkit setup wizard guides you through the installation process through a series of installation dialog boxes. Follow these steps to WebSphere Adapter Toolkit V6.0.0.1:

1. Launch the setup wizard.

Navigate to the directory where you have unzipped WebSphere Adapter Toolkit download. Double-click the file **setupwin32.exe**. The welcome panel of the setup wizard opens as shown in Figure 8-16. Click **Next** to continue.

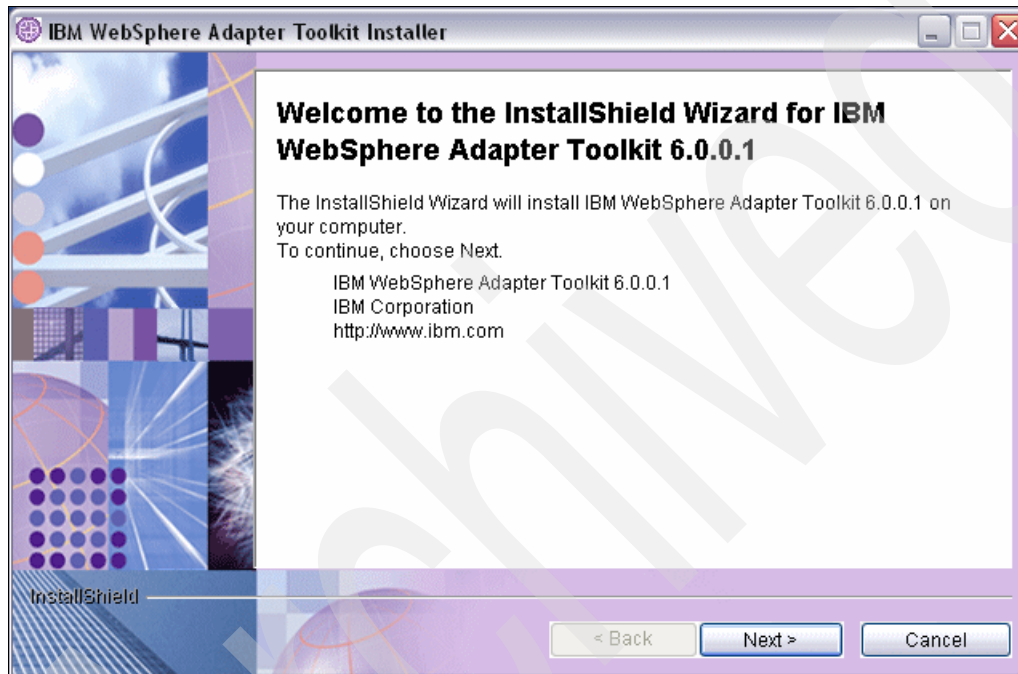


Figure 8-16 Welcome

2. The license agreement dialog box opens as shown in Figure 8-17. Review the license terms, click **I accept the license terms in the license agreement**.

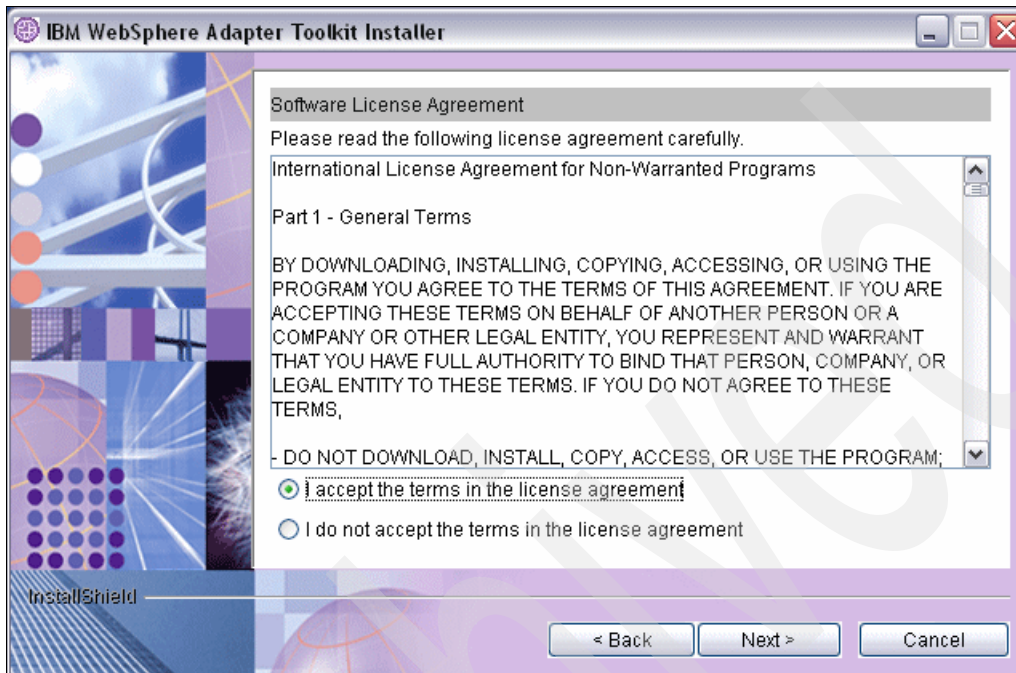


Figure 8-17 License agreement

3. On the license agreement dialog box as shown in Figure 8-17, click **Next** to continue.

4. The WebSphere Adapter Toolkit install location directory dialog box opens as shown in Figure 8-18. Click **Next** to continue and accept the default installation location directory.

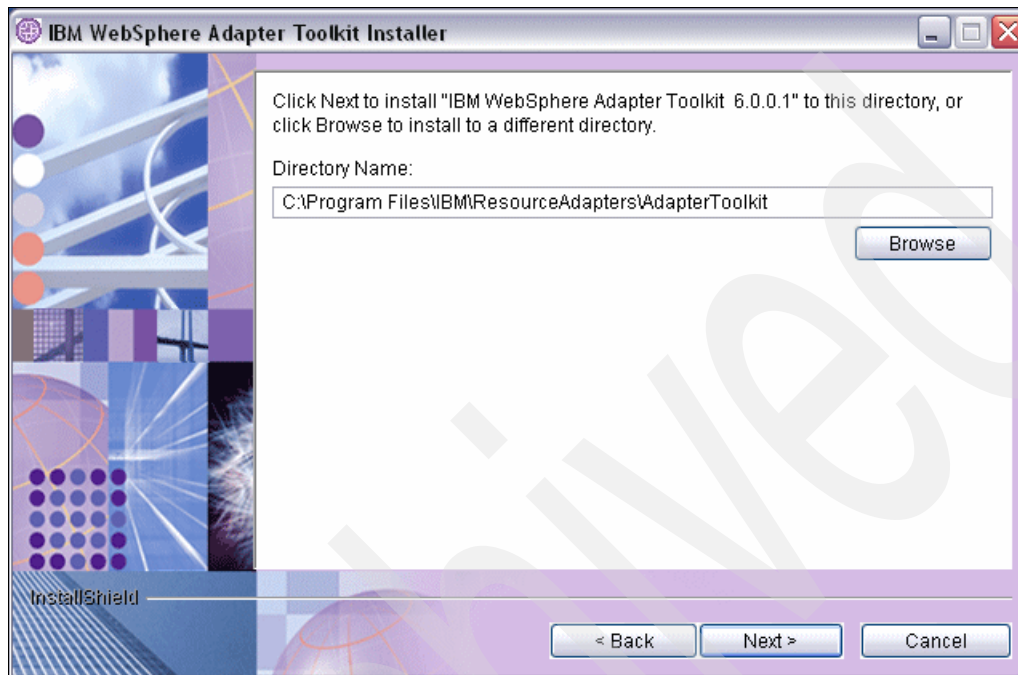


Figure 8-18 Install location directory

5. The preinstallation summary dialog box opens as shown in Figure 8-19. Review the information then click **Next** to continue.

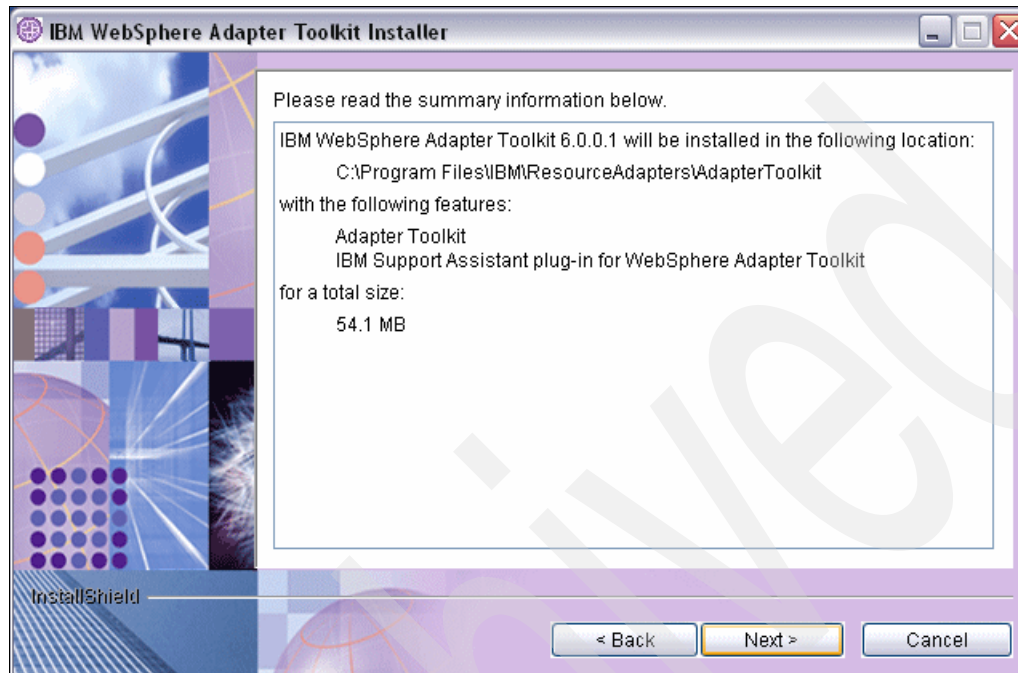


Figure 8-19 Preinstallation summary

6. The installation progress panel opens as shown in Figure 8-20. When installation is complete, the setup wizard automatically opens the next dialog box.

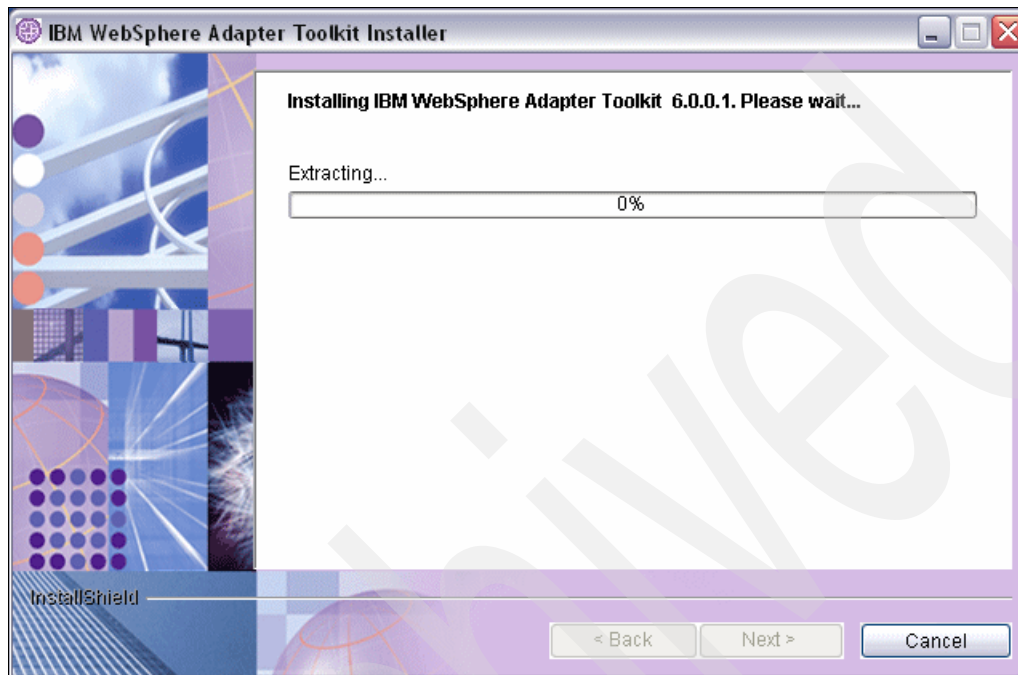


Figure 8-20 WebSphere Adapter Toolkit installation

7. The post-installation summary dialog box opens as shown in Figure 8-21. Review the summary on this panel, then click **Finish** to exit the setup wizard.

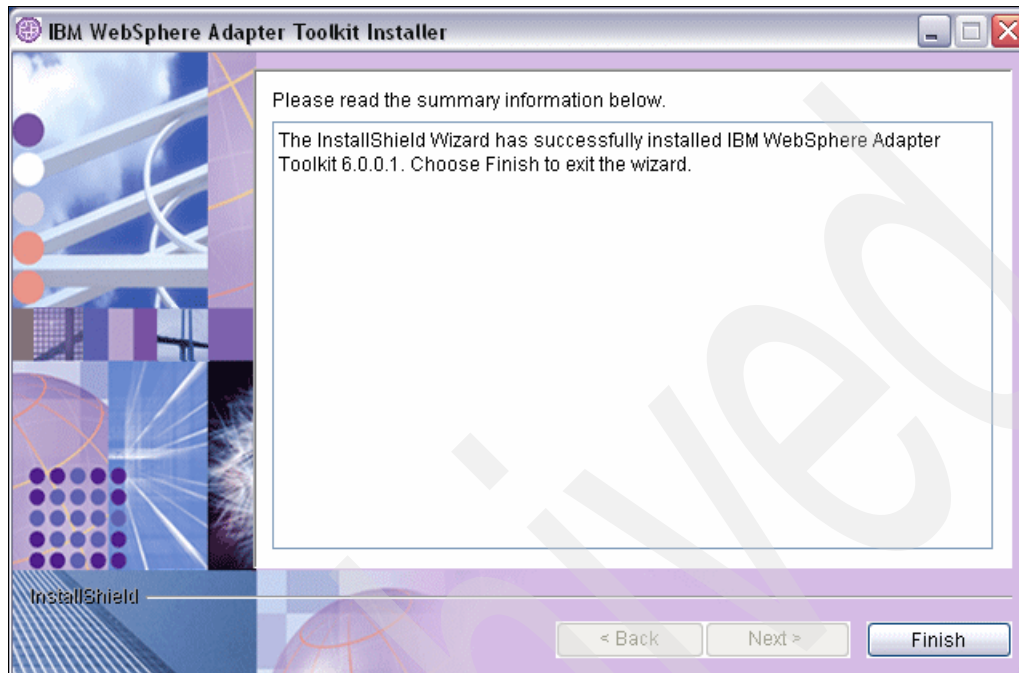


Figure 8-21 Post-installation summary

The setting up of your adapter development environment is complete.

8.4 Create a Hello World adapter

Now that adapter development environment setup is complete, we will use WebSphere Adapter Toolkit (running on WebSphere Integration Developer) to create a simple Hello World adapter. This adapter runs on WebSphere Integration Developer's test environment or WebSphere Process Server. WebSphere Process Server sends a business object to the adapter. The adapter processes this business object and returns a response business object. The response business object contains the famous Hello World greeting. Essentially this resource adapter illustrates the outbound operation of the resource adapter. Creating this Hello World sample adapter requires many steps. To understand the theory behind these steps, read Chapter 4, "Outbound request processing" on page 73. However, to keep this adapter simple, there is no underlying EIS.

Follow these simple steps to create and test the Hello World resource adapter:

1. Start **WebSphere Integration Developer**.
2. Select a new workspace as shown in Figure 8-22.

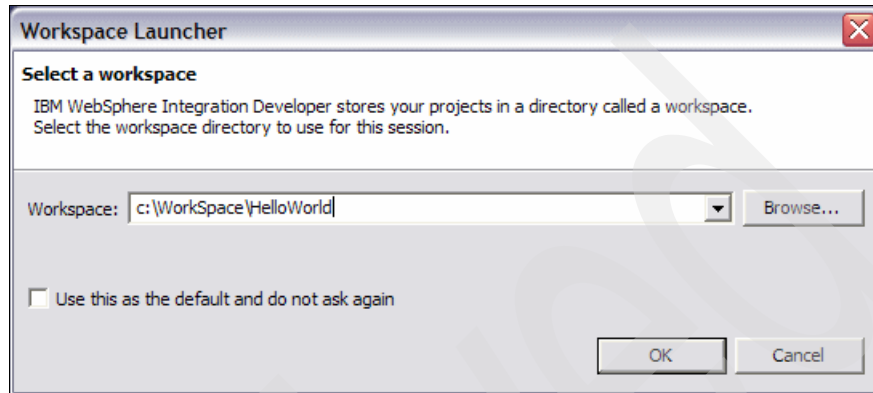


Figure 8-22 Hello World workspace

3. WebSphere Integration Developer integrated development environment opens. Close the welcome view to see the Business Integration perspective.

4. From the menu, select **File** → **New** → **Project**. This brings up the new project dialog box. See Figure 8-23.

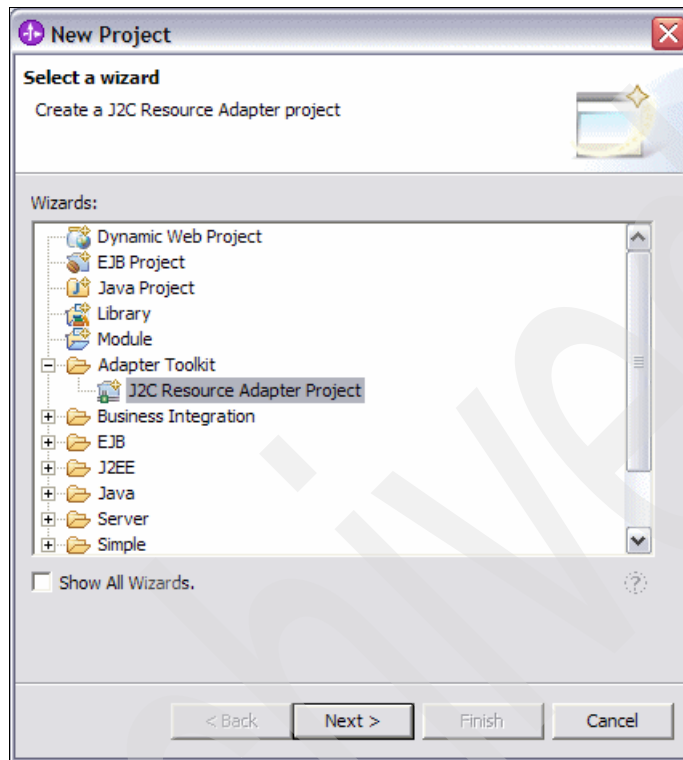


Figure 8-23 New project dialog

5. Select **J2C Resource Adapter Project** under Adapter Toolkit folder as in Figure 8-23.
6. Click **Next**.

7. Enter HelloWorld as the adapter project name as shown in Figure 8-24.

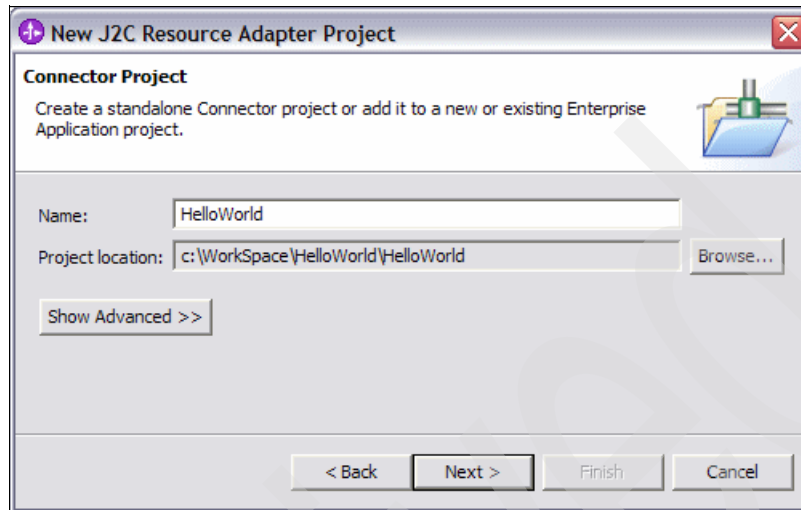
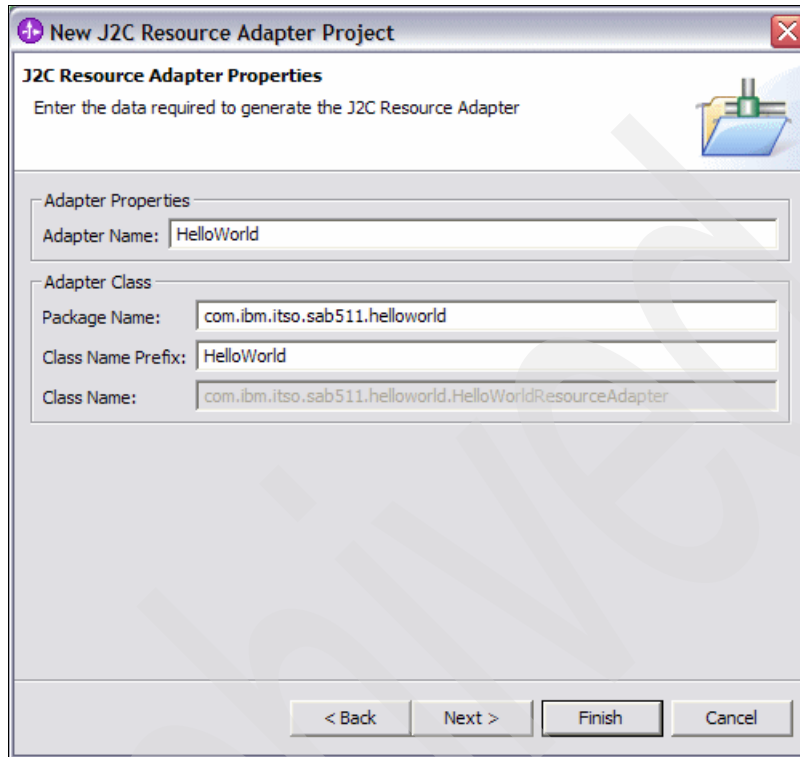


Figure 8-24 New J2C Resource Adapter Project dialog

8. Click **Next**.

9. Enter the Hello World adapter properties as shown in Figure 8-25.



The screenshot shows a Windows-style dialog box titled "New J2C Resource Adapter Project". Inside, there's a section titled "J2C Resource Adapter Properties" with the instruction "Enter the data required to generate the J2C Resource Adapter". Below this, there are two main sections: "Adapter Properties" and "Adapter Class". The "Adapter Properties" section has a single text field "Adapter Name:" containing the text "HelloWorld". The "Adapter Class" section has three text fields: "Package Name:" containing "com.ibm.itso.sab511.helloworld", "Class Name Prefix:" containing "HelloWorld", and "Class Name:" containing "com.ibm.itso.sab511.helloworld.HelloWorldResourceAdapter". At the bottom of the dialog, there are four buttons: "< Back", "Next >", "Finish", and "Cancel".

Figure 8-25 Adapter properties

10. Click **Next**.

11. Select the components to be generated as shown in Figure 8-26.

WebSphere Adapter Toolkit can generate stub classes that are required to fulfil the resource adapter side of the JCA system contracts. These stub classes are implemented by the adapter developer.

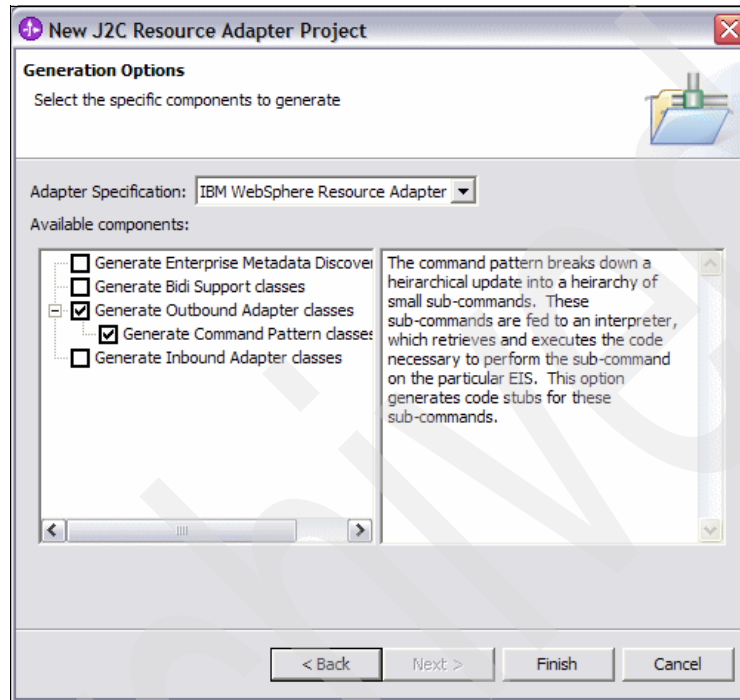


Figure 8-26 Select components to be generated by the toolkit

12. Click **Finish** to start generating selected components.

13. Click **Yes** if you are prompted to switch your perspective to J2EE as shown in Figure 8-27.

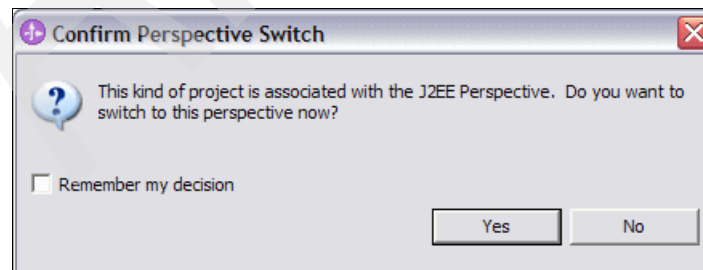


Figure 8-27 Switch to J2EE perspective

14. Expand the **connector project** folder in the project explorer view to examine the files generated by WebSphere Adapter Toolkit. See Figure 8-28.

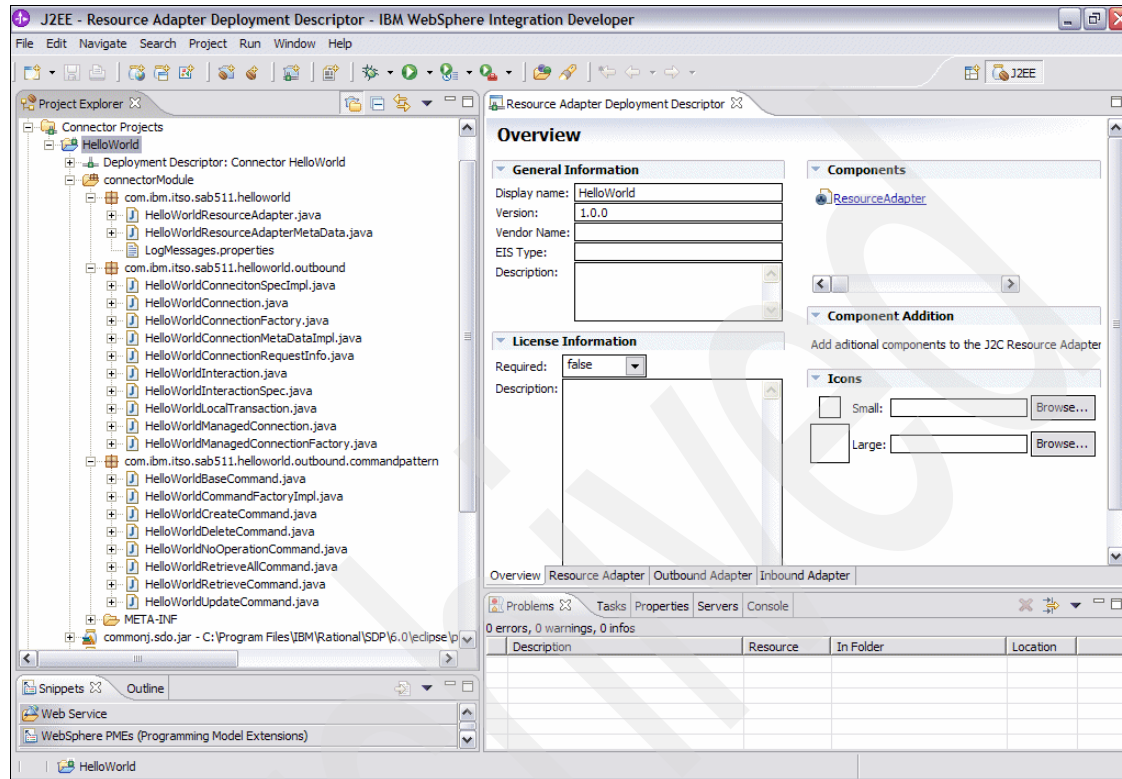


Figure 8-28 Generated files

15. Implement **getResourceAdapterMetadata** method in **HelloWorldResourceAdapter.java**. See Example 8-1.

Example 8-1 implement getResourceAdapterMetadata method

```

public com.ibm.j2ca.base.WBIResourceAdapterMetadata
getResourceAdapterMetadata()
    throws javax.resource.ResourceException {
    return new HelloWorldResourceAdapterMetaData("HelloWorld",
                                                "IBM", "1.0", false);
}

```

Note: When you perform the following steps, you might need to add additional import statements to the class you are implementing. You can use **organize import** feature of the Java editor in WebSphere Integration Developer to add the necessary imports.

16. Implement **createInteraction** method of HelloWorldConnection.java. See Example 8-2.

Example 8-2 Implement createInteraction method

```
public javax.resource.cci.Interaction createInteraction()
    throws javax.resource.ResourceException {
    return new HelloWorldInteraction(this);
}
```

17. Implement **execute** method of HelloWorldInteraction.java. See Example 8-3.

Example 8-3 Implement execute method

```
public javax.resource.cci.Record execute(
    javax.resource.cci.InteractionSpec arg0,
    javax.resource.cci.Record arg1)
    throws javax.resource.ResourceException {
    DataObject bgHello = ((WBIRRecord)arg1).getDataObject();
    DataObject boHello = bgHello.getDataObject("Hello");
    String name = boHello.getString("greeting");
    System.out.println(name);

    WBIRRecord outRecord = new WBIRRecord();
    DataObject outBG = AdapterBOUtil.copyBusinessObject(bgHello);
    DataObject outBO = outBG.getDataObject("Hello");
    outBO.setString("greeting", "HelloWorld from: " + name);
    outRecord.setDataObject(outBG);

    return outRecord;
}
```

18. Implement **getWBICConnection** method of HelloWorldManagedConnection.java. See Example 8-4.

Example 8-4 Implement getWBICConnection method

```
public java.lang.Object getWBICConnection(
    javax.resource.spi.security.PasswordCredential arg0, boolean arg1)
    throws javax.resource.ResourceException {
    return new HelloWorldConnection(this);
}
```

19. Implement **createManagedConnection** method of HelloWorldManagedConnectionFactory.java. See Example 8-5.

Example 8-5 Implement createManagedConnection method

```
public javax.resource.spi.ManagedConnection createManagedConnection(
    javax.security.auth.Subject arg0,
    javax.resource.spi.ConnectionRequestInfo arg1)
    throws javax.resource.ResourceException {
    return new HelloWorldManagedConnection(this, arg0,
                                           (WBICConnectionRequestInfo)arg1);
}
```

20. Implement **createConnectionFactory** method of HelloWorldManagedConnectionFactory.java. See Example 8-6.

Example 8-6 Implement createConnectionFactory method

```
public java.lang.Object createConnectionFactory(
    javax.resource.spi.ConnectionManager arg0)
    throws javax.resource.ResourceException {
    return new HelloWorldConnectionFactory(arg0, this);
}
```

21. Add the **functionName** getter and setter methods to HelloWorldInteractionSpec.java. See Example 8-7.

Example 8-7 Function name getter and setter methods

```
public String getFunctionName() {
    return super.getFunctionName();
}

public void setFunctionName(String arg0) {
    super.setFunctionName(arg0);
}
```

22. Switch to the **Business Integration** perspective.

23. Right-click anywhere inside the **Business Integration** view to open the **context** menu. See Figure 8-29.

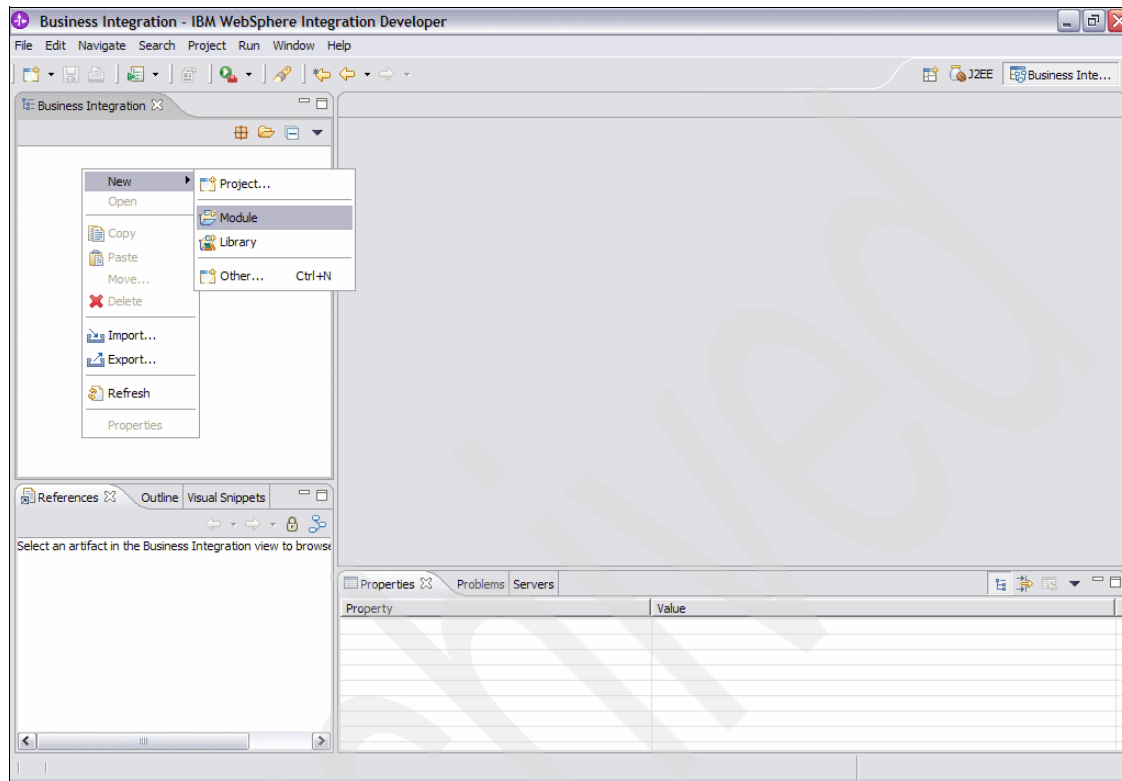


Figure 8-29 Business Integration view's context menu

24. Click **New** → **Module** as shown in Figure 8-29.

25. Enter HelloWorldModule as shown in Figure 8-30.

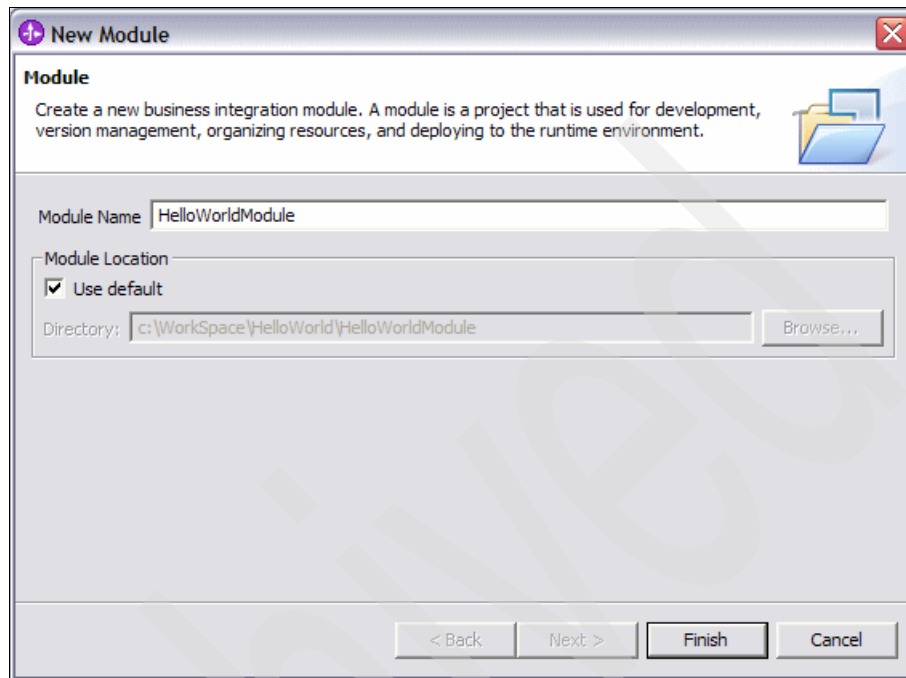


Figure 8-30 New Module dialog

26. Click **Finish**.

27. Right-click **HelloWorldModule** in the Business Integration view.

28. Select **Open Dependency Editor** menu option.

29. Inside Dependency Editor, click **J2EE**.

30. Click **Add**. See Figure 8-31.

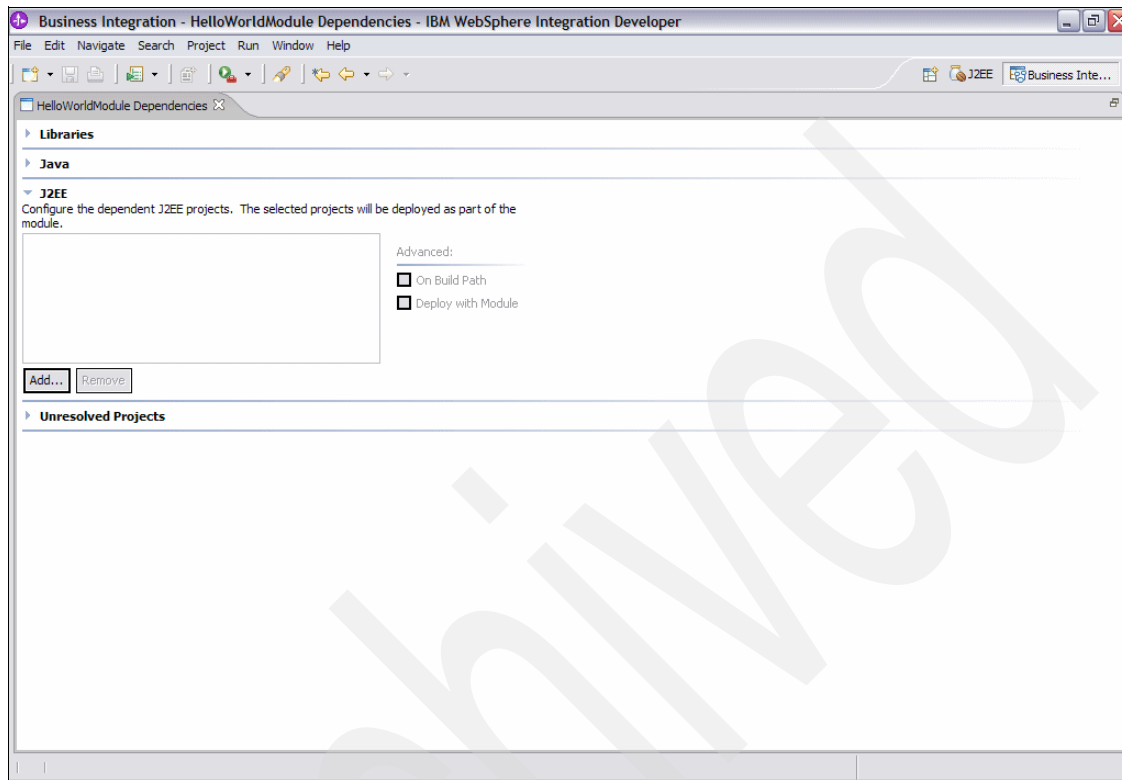


Figure 8-31 Dependency editor

31. Select **HelloWorld**. See Figure 8-32.

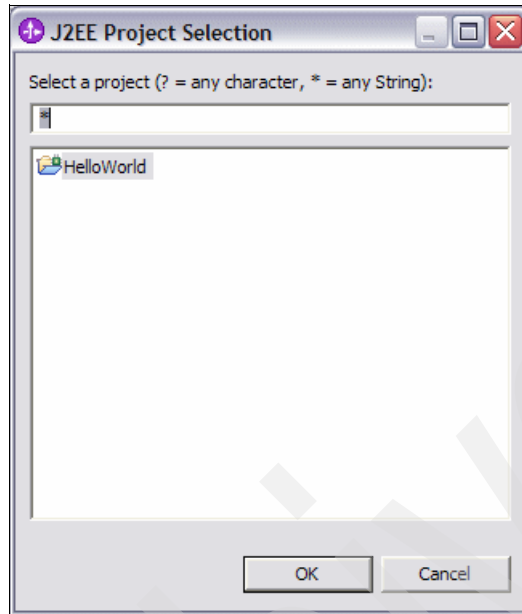


Figure 8-32 Select dependent J2EE project

32. Click **OK**.

33. In Business Integration view, right-click **Data Types** → **New** → **Business Object**. See Figure 8-33.

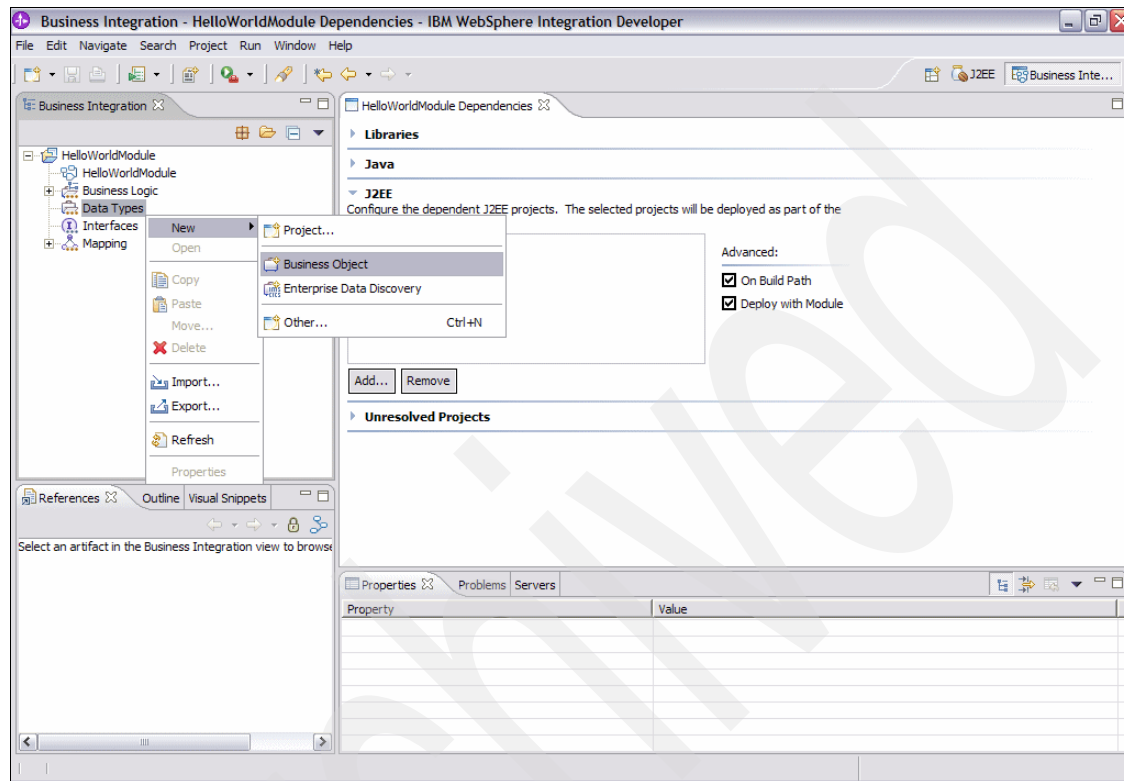


Figure 8-33 Open new business object dialog

34. Enter Hello in the name field of the new business object dialog box.
35. Click **Finish**.

36. In the business object editor, select the **icon** as shown in Figure 8-34 to add a new greeting attribute.

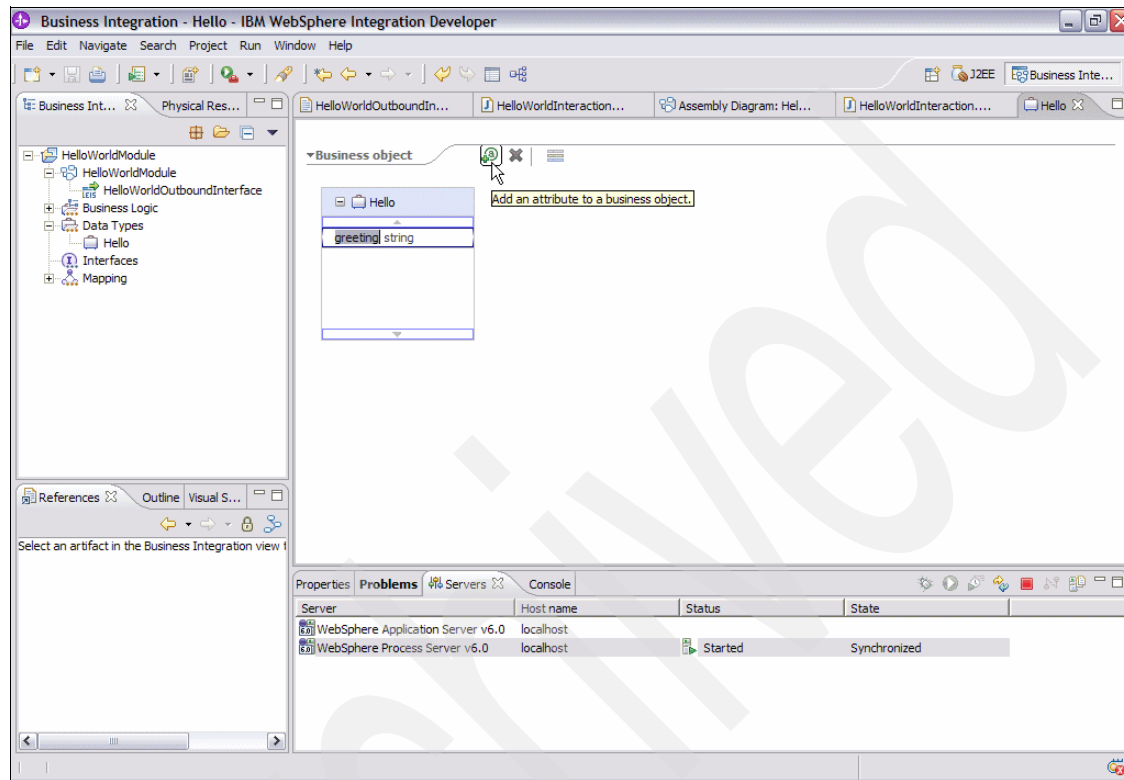


Figure 8-34 Add name attribute to Person business object

37. Save Hello business object.

38. Right-click **Hello business object** → **Create a Business Graph** as shown in Figure 8-35.

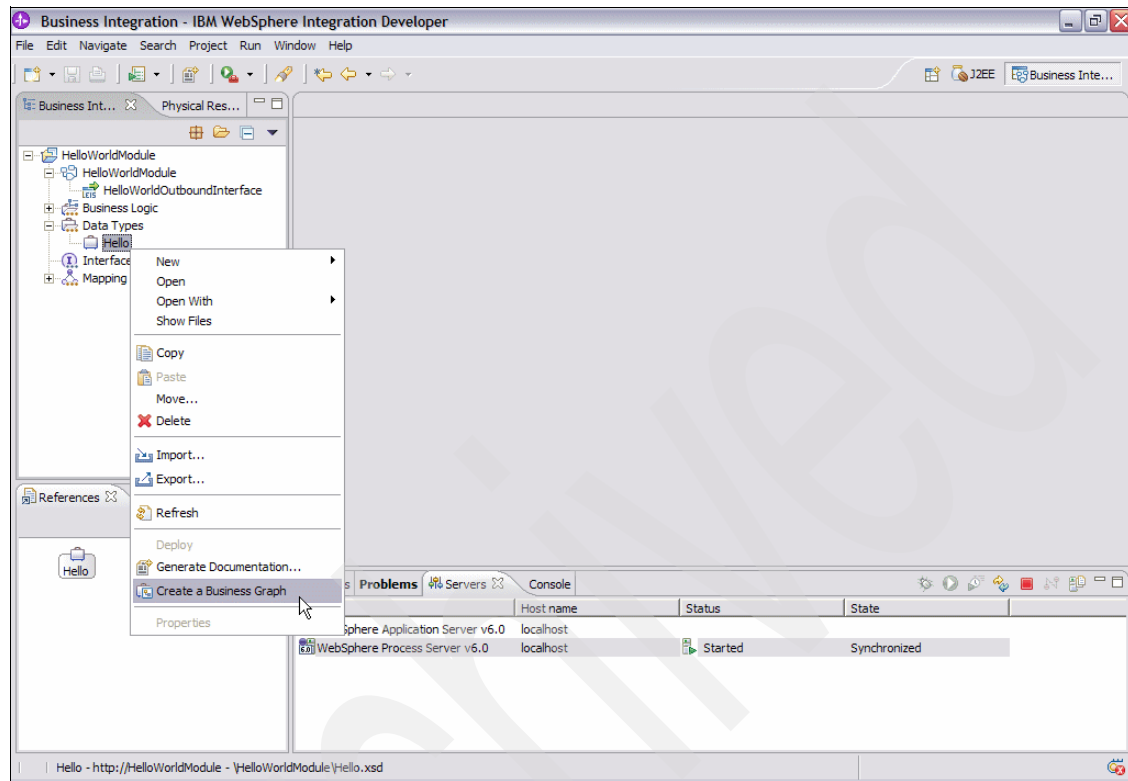


Figure 8-35 Generate business graph

39. In the Business Integration view, right-click **Interfaces** → **New** → **Interface** as shown in Figure 8-36.

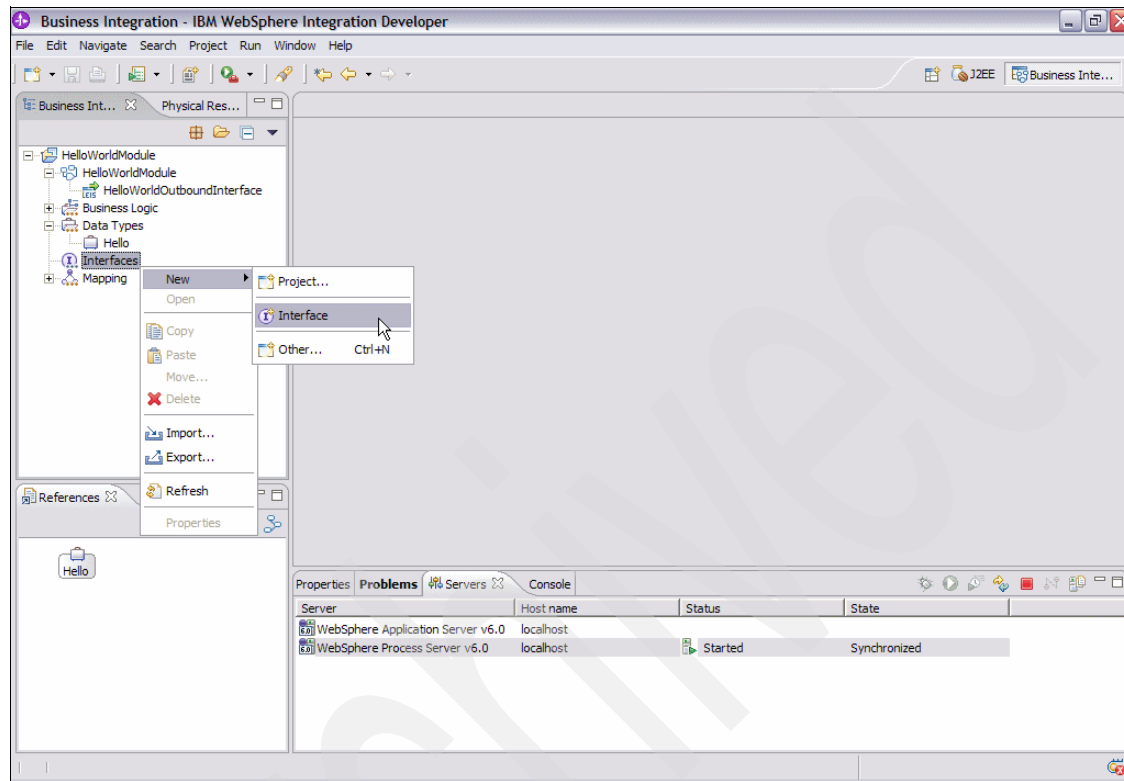
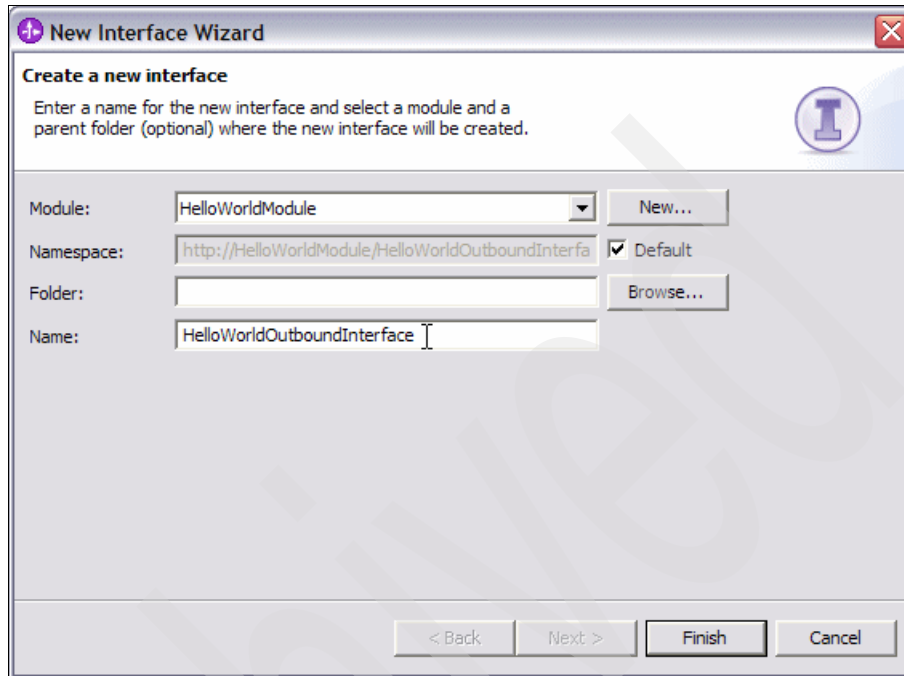


Figure 8-36 Create new interface

40. Enter the name of the outbound interface as shown in Figure 8-37.



The image shows a 'New Interface Wizard' dialog box. The title bar says 'New Interface Wizard'. Below the title bar, there is a section titled 'Create a new interface' with a sub-instruction: 'Enter a name for the new interface and select a module and a parent folder (optional) where the new interface will be created.' To the right of this text is a blue circular icon with a white 'I'. The main area of the dialog contains four input fields: 'Module:' with a dropdown menu showing 'HelloWorldModule' and a 'New...' button; 'Namespace:' with a text box containing 'http://HelloWorldModule/HelloWorldOutboundInterfa' and a checked 'Default' checkbox; 'Folder:' with an empty text box and a 'Browse...' button; and 'Name:' with a text box containing 'HelloWorldOutboundInterface'. At the bottom of the dialog are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

Figure 8-37 Enter new interface name

41. Click **Finish**.

42. Click **Add Request Response** as shown in Figure 8-38 to create a new outbound operation.

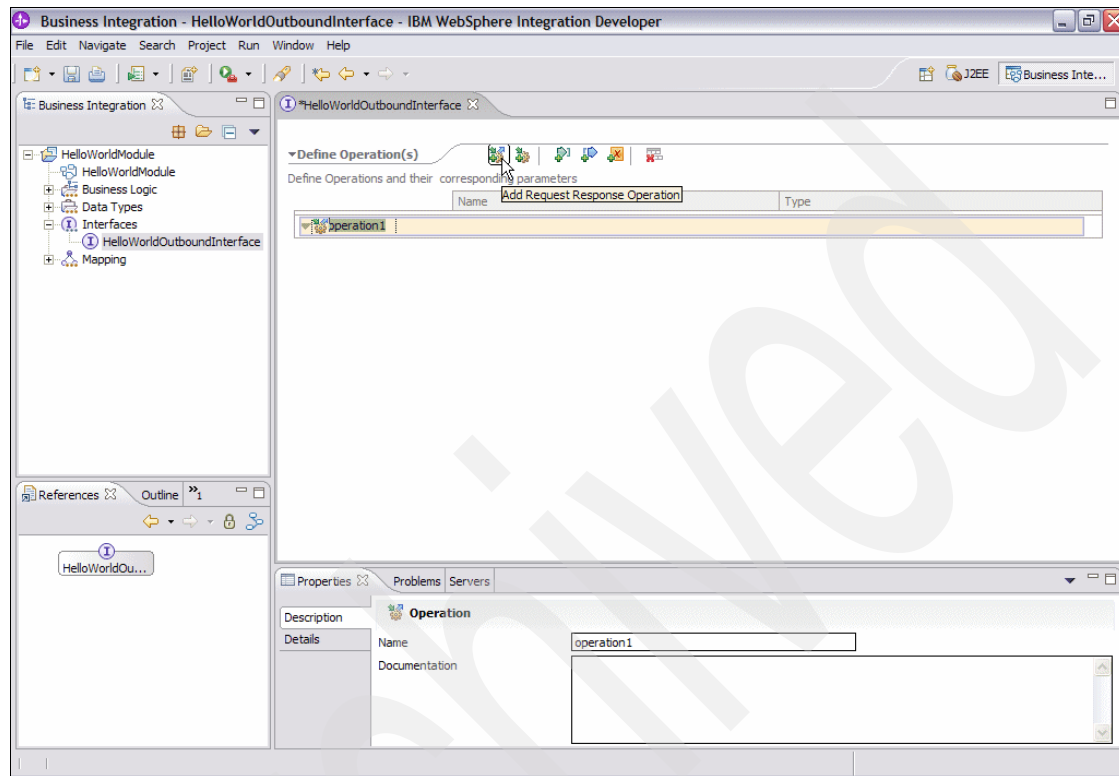


Figure 8-38 Create new outbound operation

43. Rename interface method operation1 to sayGreetings.

44. With **sayGreetings** operation selected, click **Add Input** as shown in Figure 8-39 to add an input parameter.

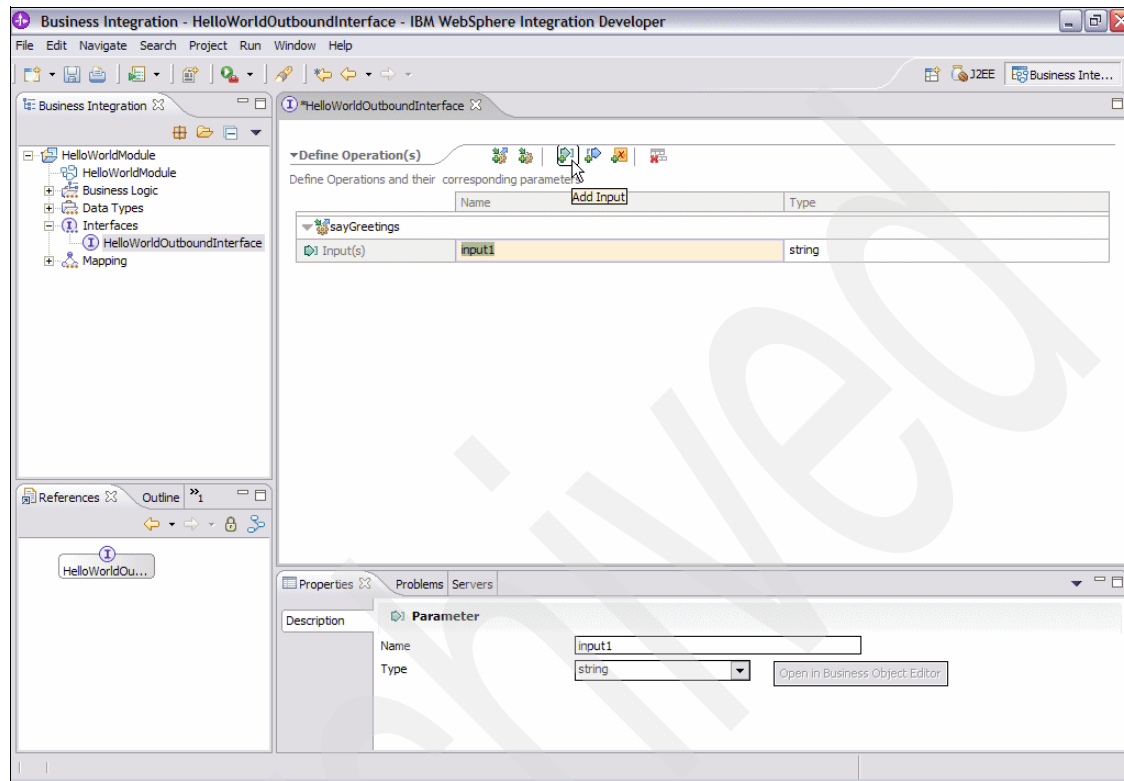


Figure 8-39 Add input parameter to outbound operation

45. Rename input1 to inputHelloBG.

46. Change input type from string to HelloBG.

47. Select **Add Output** as shown in Figure 8-40.

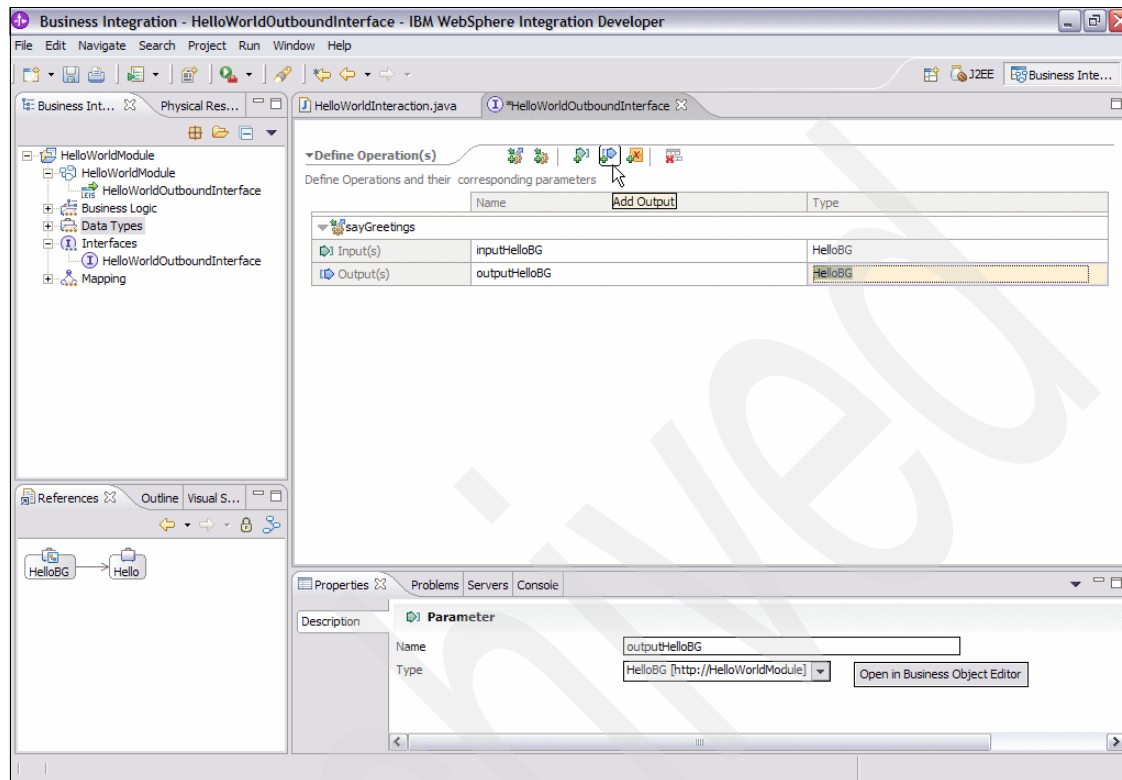


Figure 8-40 Add output parameter to outbound operation

48. Replace parameter name output1 with outputHelloBG.

49. Change output type from string to HelloBG.

50. From the menu click **File** → **Save All**.

8.4.1 Service Component Architecture import for outbound operation

WebSphere Process Server uses a Service Component Architecture programming model. In order for a resource adapter to plug into SCA and be made available for invocation (for outbound operation) by other SCA components, it must be configured as an SCA EIS import. SCA import is an abstract concept. We use Web Service WSDL and SCDL files to bind adapter implementation to the SCA programming model.

The next few steps show you how to create an SCDL import file that binds the resource adapter to SCA:

1. From the menu, click **Window** → **Show View** → **Physical Resources** to open the Physical Resource view.
2. Hit **Ctrl+N** keys to open the new wizard.
3. Select **Simple** → **File** as shown in Figure 8-41.

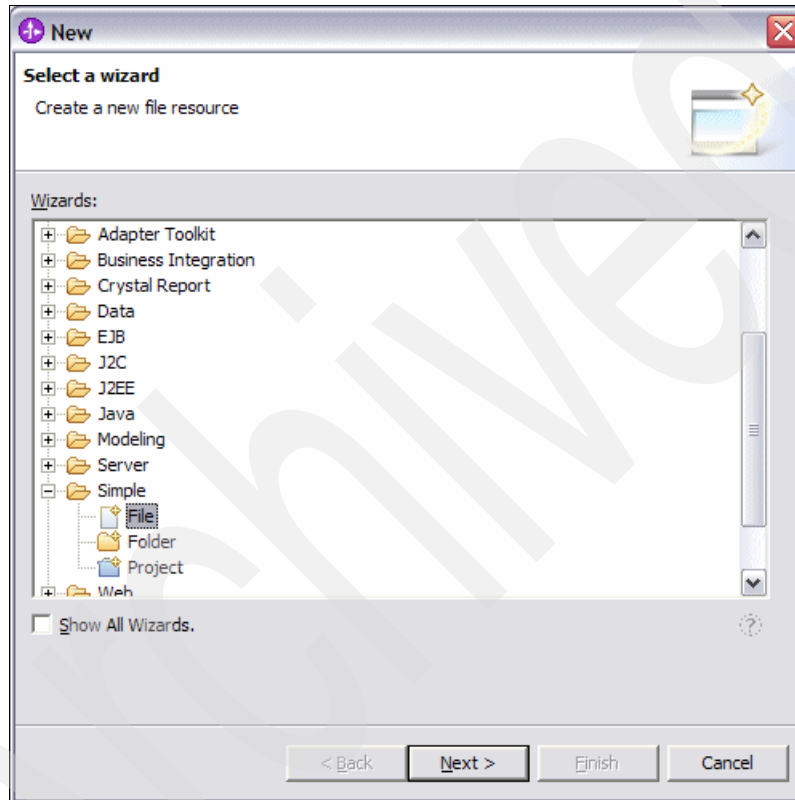


Figure 8-41 Create a new file

4. Click **Next**.
5. Select **HelloWorldModule**.
6. Enter `HelloWorldOutboundInterface.import` as the new file name.
7. Click **Finish**.

- Copy the code from Example 8-8 into `HelloWorldOutboundInterface.import` file.

Example 8-8 HelloWorldOutboundInterface.import

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:import xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:eis="http://www.ibm.com/xmlns/prod/websphere/scdl/eis/6.0.0"
  xmlns:ns1="http://HelloWorldModule/HelloWorldOutboundInterface"
  xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/6.0.0"
  xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/6.0.0"
  displayName="HelloWorldOutboundInterface"
  name="HelloWorldOutboundInterface">
  <interfaces>
    <interface xsi:type="wsdl:WSDLPortType"
portType="ns1:HelloWorldOutboundInterface">
      <method name="sayGreetings"/>
    </interface>
  </interfaces>
  <esbBinding xsi:type="eis:EISImportBinding"
    dataBindingType="com.ibm.j2ca.extension.emd.runtime.WBIDataBindingImpl">
    <resourceAdapter name="HelloWorldModuleApp.HelloWorld"
      type="com.ibm.itso.sab511.helloworld.HelloWorldResourceAdapter"/>
    <connection
type="com.ibm.itso.sab511.helloworld.outbound.HelloWorldManagedConnectionFactory"
y"
interactionType="com.ibm.itso.sab511.helloworld.outbound.HelloWorldInteractionSpec">
      <authentication resAuthAlias="widNode/Red_Alias"/>
    </connection>
    <methodBinding method="sayGreetings">
      <interaction>
        <properties>
          <functionName>Greeting</functionName>
        </properties>
      </interaction>
    </methodBinding>
  </esbBinding>
</scdl:import>
```

9. From the menu click **File** → **Save All**.

10. Click **HelloWorldOutboundInterface** as shown in Figure 8-42. The HelloWorld adapter import icon is displayed in the assembly diagram.

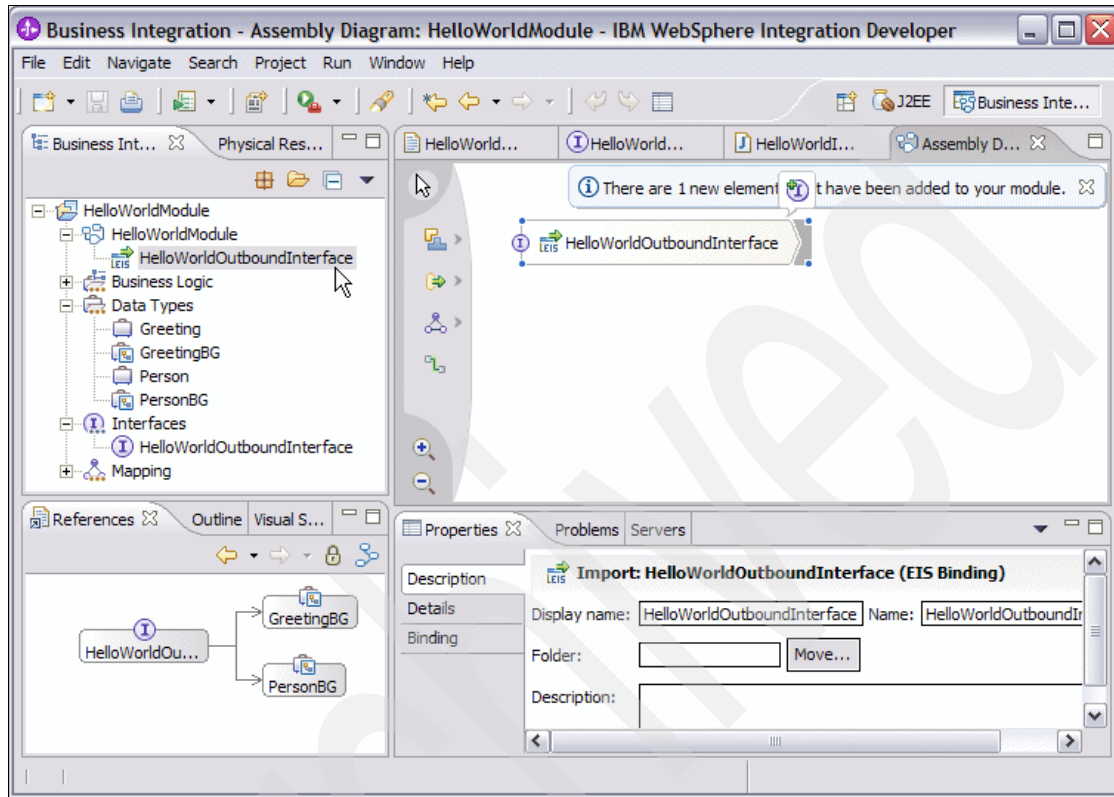


Figure 8-42 HelloWorld adapter import

11. From the menu, click **Project** → **Clean...**
12. Click **OK** in the clean dialog box.
13. Switch to the **J2EE** perspective.

8.4.2 Export application EAR file

Export application EAR file from your development environment:

1. Switch to the **J2EE** perspective.
2. Open the **HelloWorldModuleApp** deployment descriptor.

3. Select **Module** to ensure Connector HelloWorld.rar file is included as part of the application modules. See Figure 8-43.

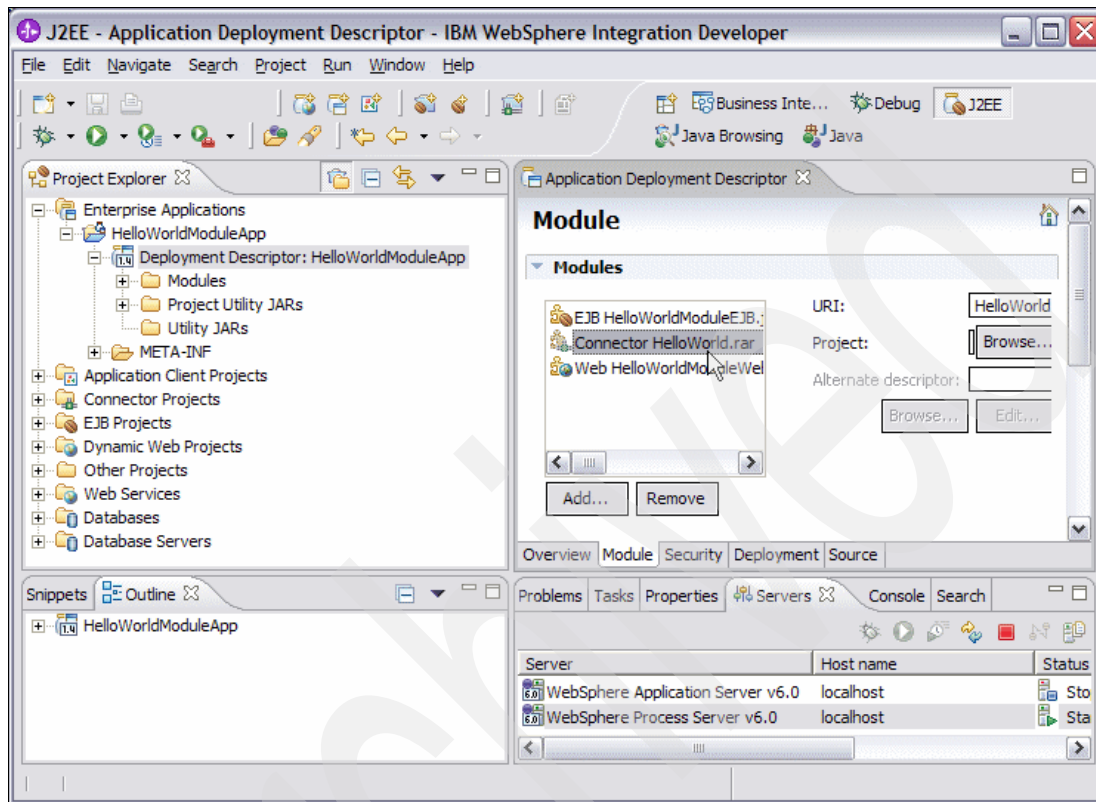


Figure 8-43 HelloWorldModuleApp deployment descriptor

4. If Connector HelloWorld.rar module is not included, click **Add** to add the HelloWorld connector project to the HelloWorldModuleApp application.
5. Right-click **HelloWorldModuleApp** under the Enterprise Applications folder.
6. Select **Export** → **EAR file**.
7. Export the **HelloWorld** application to a location on the file system.

8.4.3 Dependent files

Dependent files are normally packaged inside the resource adapter RAR file. Our Hello World adapter depends on WebSphere adapter foundation classes JAR file. To avoid discussing adapter packaging in this section, we simply put this dependent JAR file in the \lib\ directory of WebSphere Process Server. The

dependent files must be packaged with the adapter RAR file in a production environment:

1. Copy dependent jar file from:

C:\Program
Files\IBM\ResourceAdapters\AdapterToolkit\adapter\base\lib\CWYBS_AdapterFou
ndation.jar

to:

<WebSphere Integration Developer install location>\runtimes\bi_v6\lib

2. Start **WebSphere Process Server V6.0** from the server view.
3. Right-click **WebSphere Process Server V6.0** in the server view to open the **context** menu.
4. Select **Run administrative console**. See Figure 8-44.

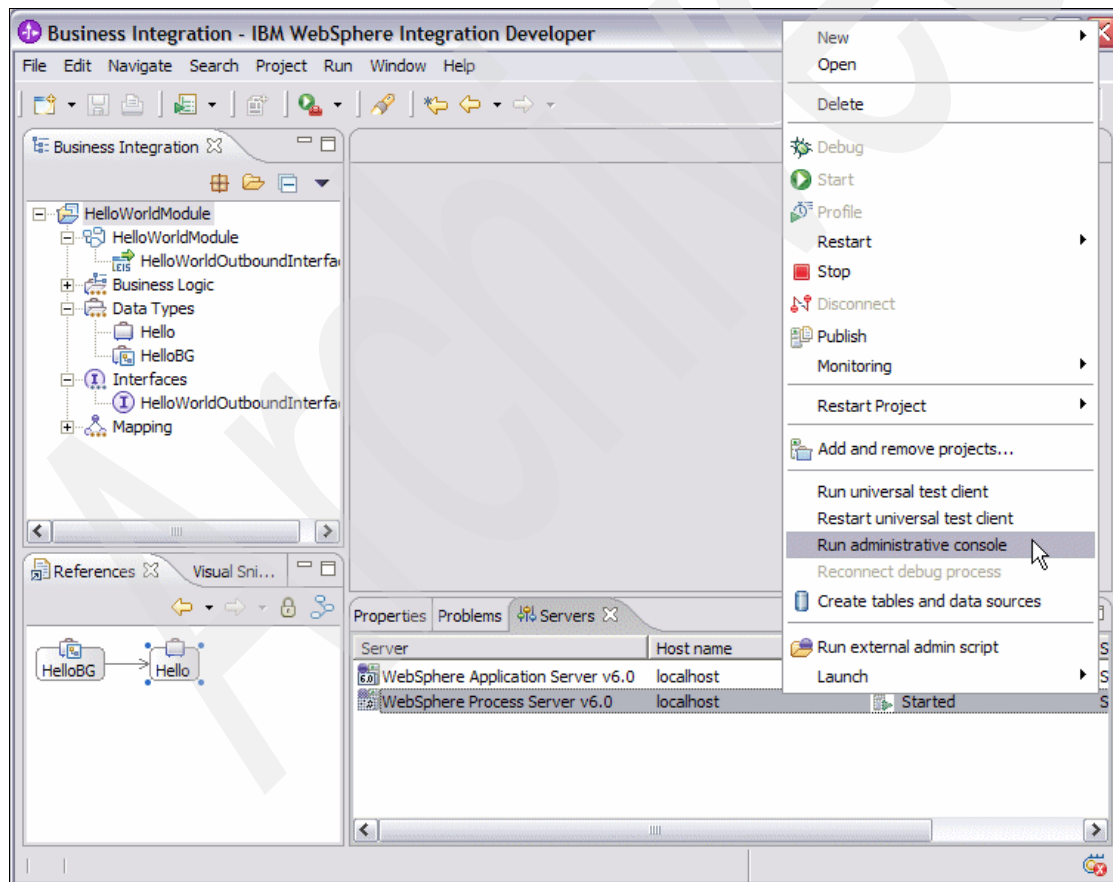


Figure 8-44 Open WebSphere Process Server administrative console

5. Click **Log in** in the Administrative console view to enter WebSphere Process Server's administrative console.

8.4.4 Configure J2C authentication alias

In order to deploy and start the adapter successfully on WebSphere Process Server, we need to configure J2C authentication alias in both places:

- ▶ In our HelloWorldOutboundInterface.import file:

```
<authentication resAuthAlias="widNode/Red_Alias"/>
```
- ▶ On WebSphere Process Server using its administration console.

This is needed even our HelloWorld adapter does not require any security. This requirement has been removed in the next version of WebSphere Adapter Foundation Classes.

1. Click **Security** → **Global** security in Administrative console.
2. Click **JAAS Configuration** on the right.

3. Click **J2C Authentication data** as shown in Figure 8-45.

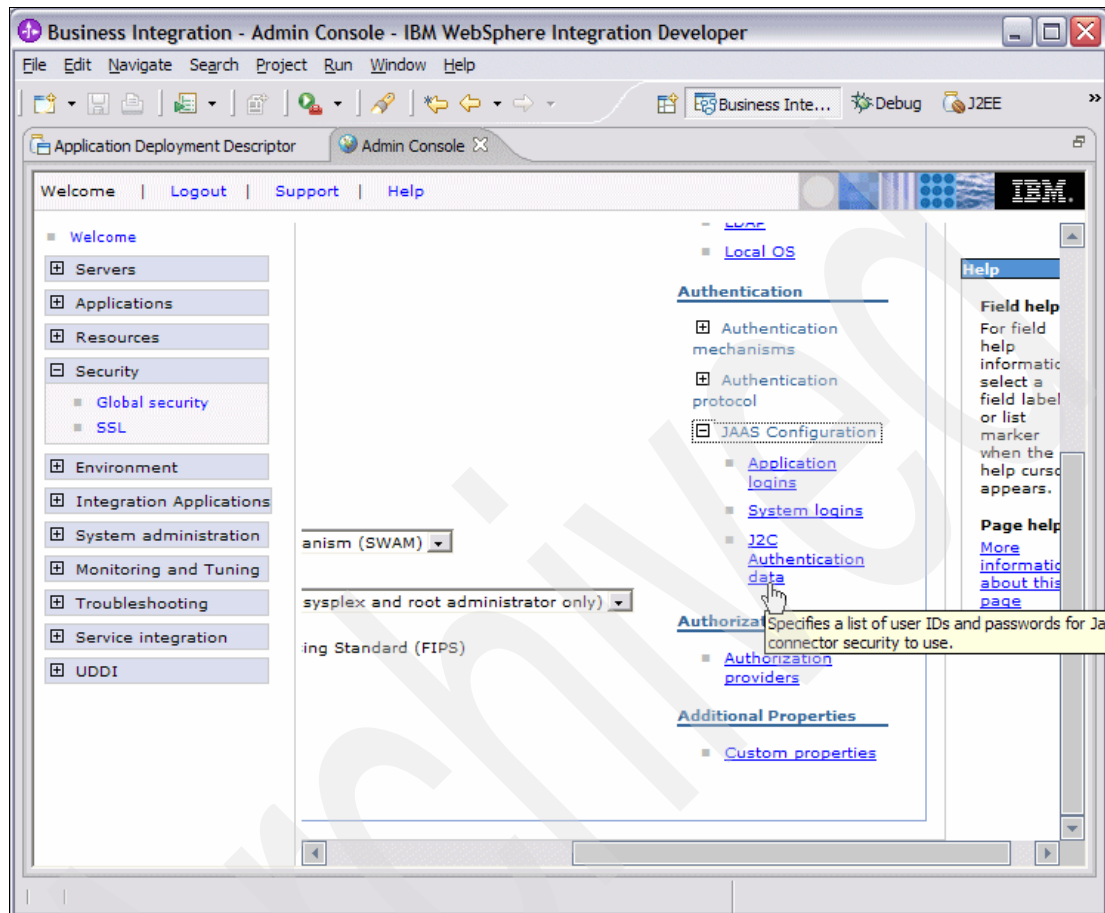


Figure 8-45 Configure J2C Authentication data

4. Click **New** to add a new J2C authentication alias. See Figure 8-46.

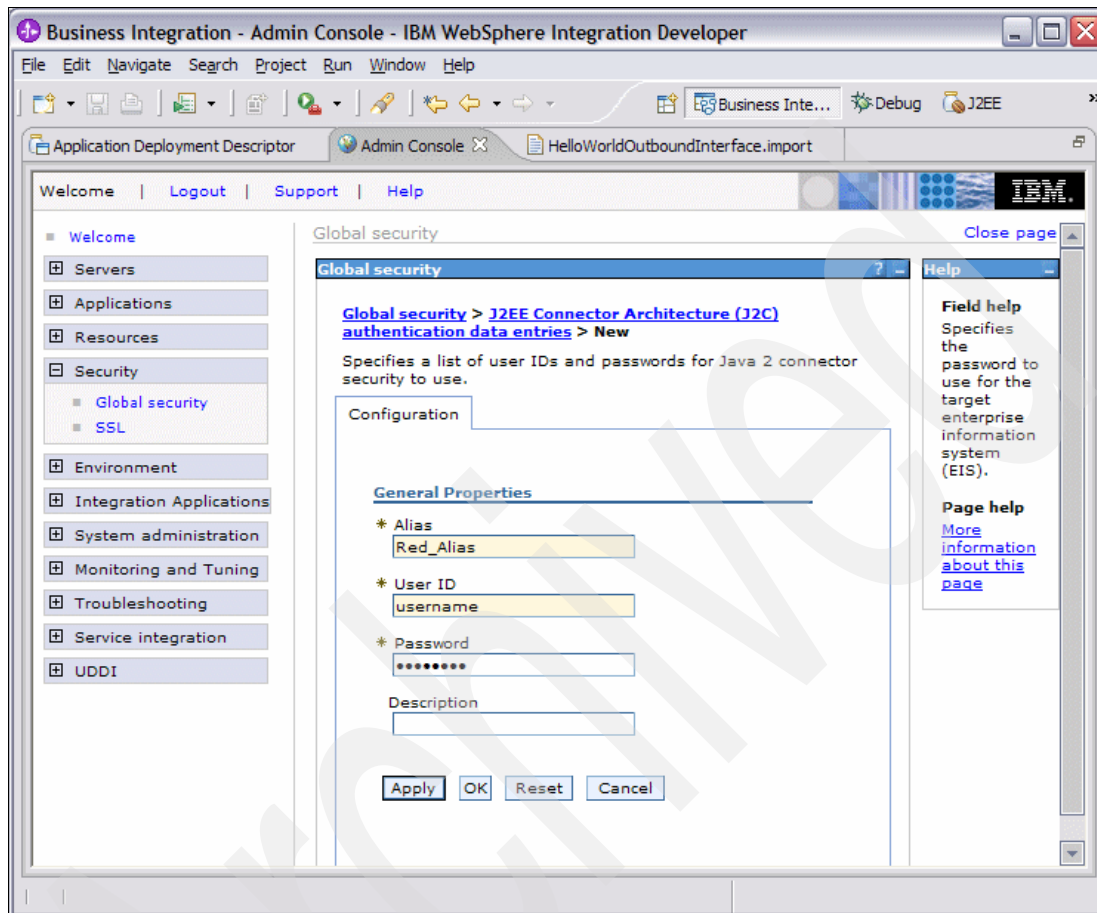


Figure 8-46 Create new authentication alias

5. Enter Red_Alias as the new alias.

This alias must match the alias in HelloWorldOutboundInterface.import file:

```
<authentication resAuthAlias="widNode/Red_Alias"/>
```

6. Enter a user name and password to connect to the EIS.

Because there is no underlying HelloWorld EIS, you can enter any user name and password.

8.4.5 Install the HelpWorld application

1. Click **Applications** → **Install New Application** in Admin console view.
2. Click **Browse** to select the application EAR file you saved in step 7 on page 205 under 8.4.2, “Export application EAR file” on page 204. See Figure 8-47.

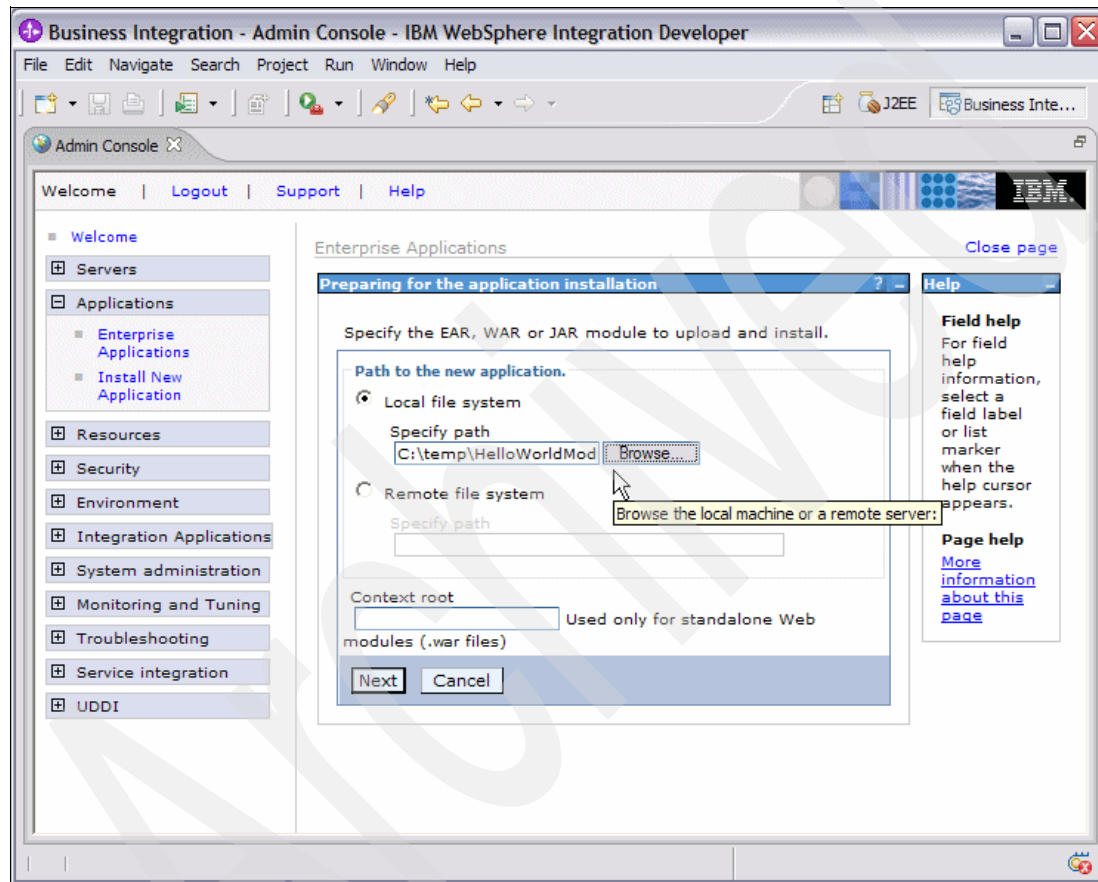


Figure 8-47 Install Hello World application to WebSphere Process Server

3. Click **Next**.
4. Click **Next**.
5. Click **Summary Step**.

6. Click **Finish**. See Figure 8-48.

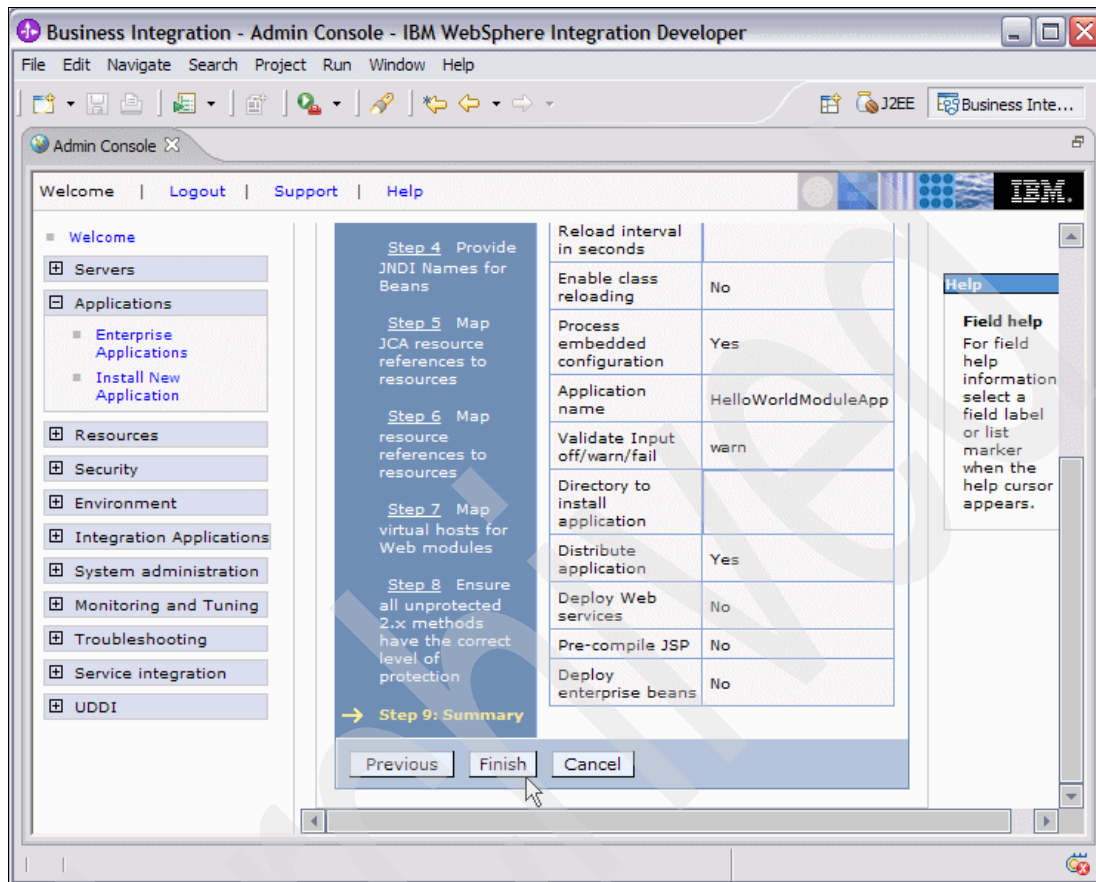


Figure 8-48 Install Hello World application (continued)

7. Click **Save to Master Configuration**.
8. Click **Save**.
9. On the left pane of the Admin console, click **Applications** → **Enterprise Applications** to show the installed applications.

10. Select **HelloWorldModuleApp** as shown in Figure 8-49.

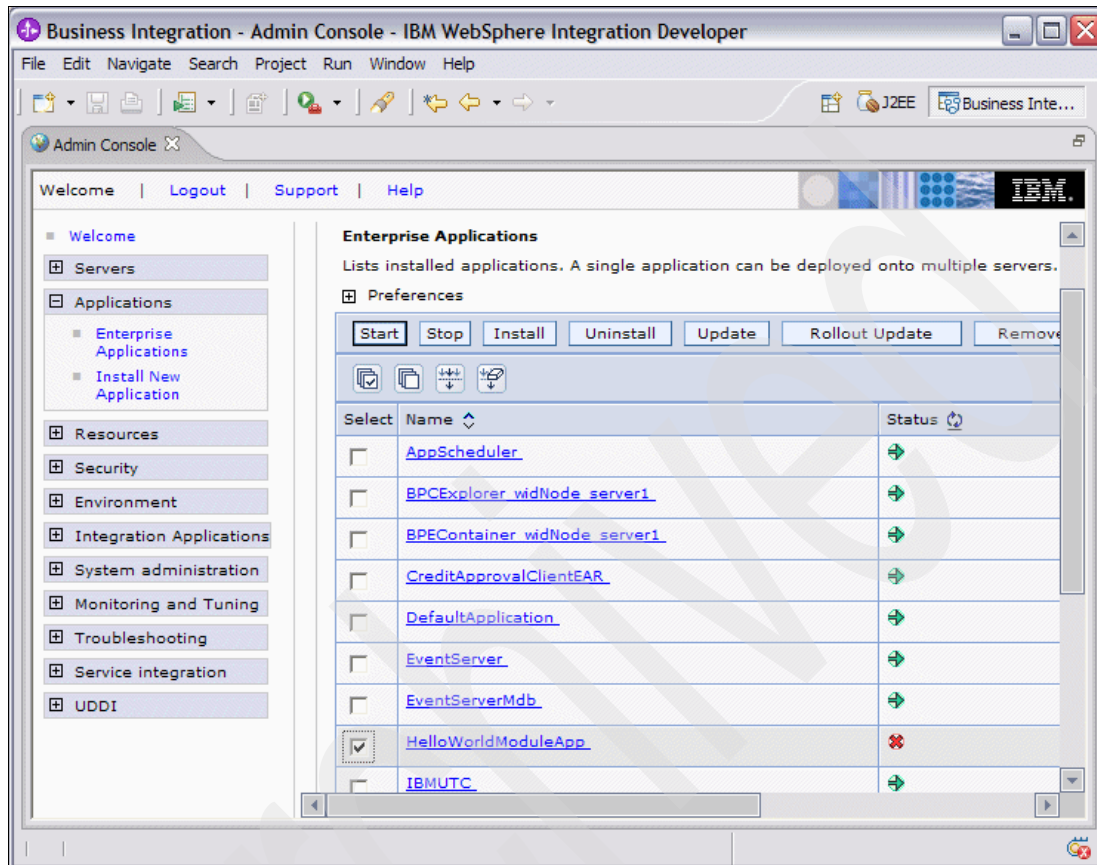


Figure 8-49 Start Hello Wold application

11. Click **Start**.

8.4.6 Configure server

To configure the server, perform the following steps:

1. Double-click **WebSphere Process Server v.6** in the servers view.
2. Select **Run server with resource on Server**.

8.4.7 Test HelloWorld adapter

To test the HelloWorld adapter, follow these steps:

1. Switch to the **Business Integration** perspective.
2. Open the **Assembly Diagram editor**.
3. Right-click **HelloWorldOutboundInterface** EIS import.
4. Click **Test Component**, as Figure 8-50 shows.

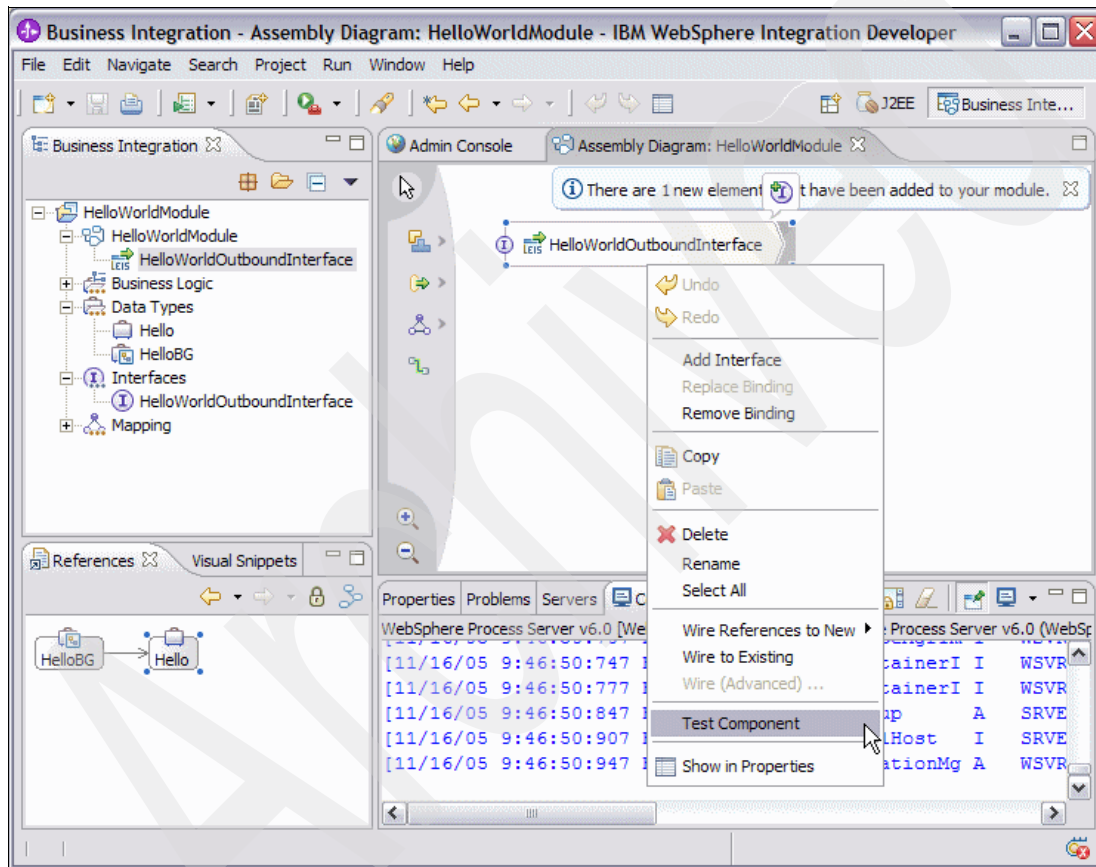


Figure 8-50 Open Test Component tool to test Hello World adapter

5. Enter a name for the greeting attribute of Hello business object as shown in step 7 under 8.4.7, “Test HelloWorld adapter” on page 213. Enter parameters for the request business objects as Figure 8-51 shows.

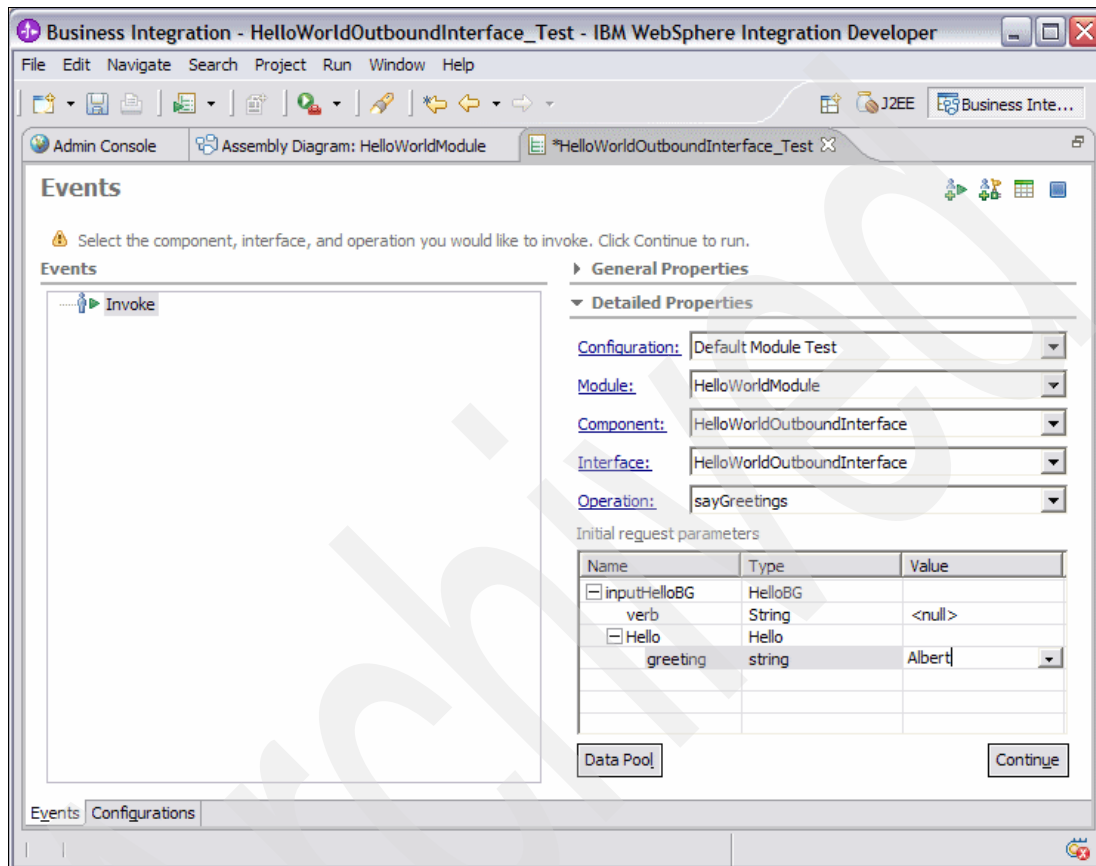


Figure 8-51 Enter parameters for the request business object

6. Click **Continue** to emulate sending a business object from WebSphere Process Server to Hello World adapter.
7. Select **WebSphere Process Server V6.0** as your deployment location.
8. Click **Finish**.

9. Examine the business object returned by Hello World adapter to WebSphere Process Server. The greeting attribute of this business object contains the famous Hello World message. See Figure 8-52.

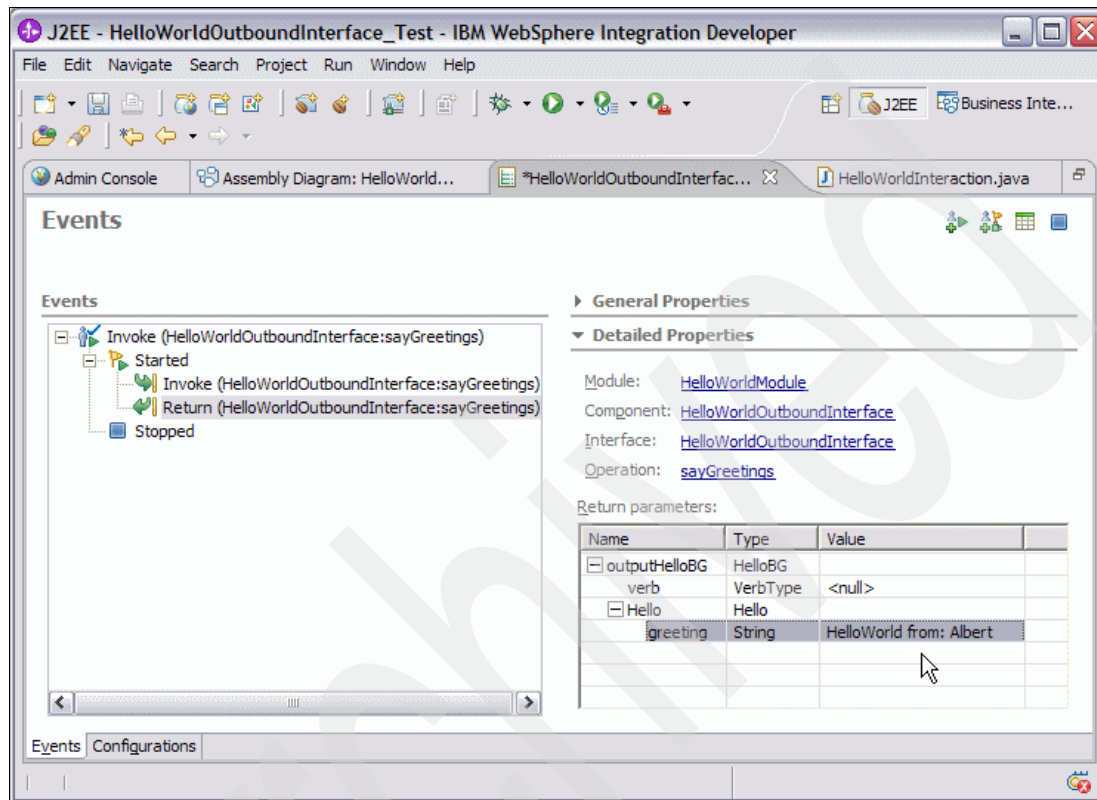


Figure 8-52 Response from HelloWorld adapter

8.5 Importing a sample adapter into your development environment

To import a sample adapter into your development environment, perform the following steps:

1. Start **WebSphere Integration Developer**.
2. From the menu, click **File** → **Import** to open the Import dialog box.
3. Click **Project Interchange** → **Next** to open the **Import Projects** dialog box.

4. On the From zip file text field, enter:

C:\Program
Files\IBM\ResourceAdapters\AdapterToolkit\adapter\twineball\src\CWYAT_Twine
Ball_src.zip

5. Select **TwineBall Sample** as shown in Figure 8-53.

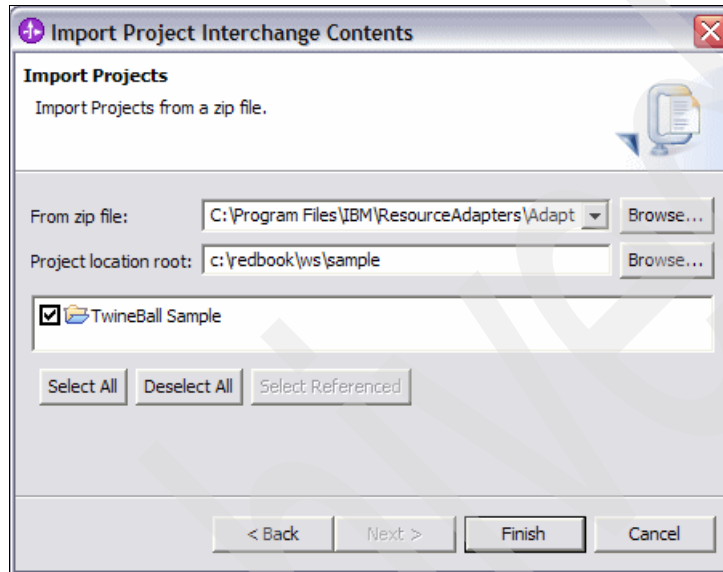


Figure 8-53 Import TwineBall Sample

6. A sample adapter called TwineBall has been imported into a connector project under Connector Project folder in your J2EE perspective.
7. In the Problems view, you see a number of errors. We need to update the project's build path to resolve these external library dependencies. Follow these steps to add dependent libraries to the build path:
 - a. Right-click the **TwineBall Sample** connector project in your J2EE perspective, select **properties** to open **Properties for TwineBall Sample** dialog box.

- b. From Properties for TwineBall Sample dialog box, select **Java Build Path**, then select the **Libraries** tab, as Figure 8-54 shows.

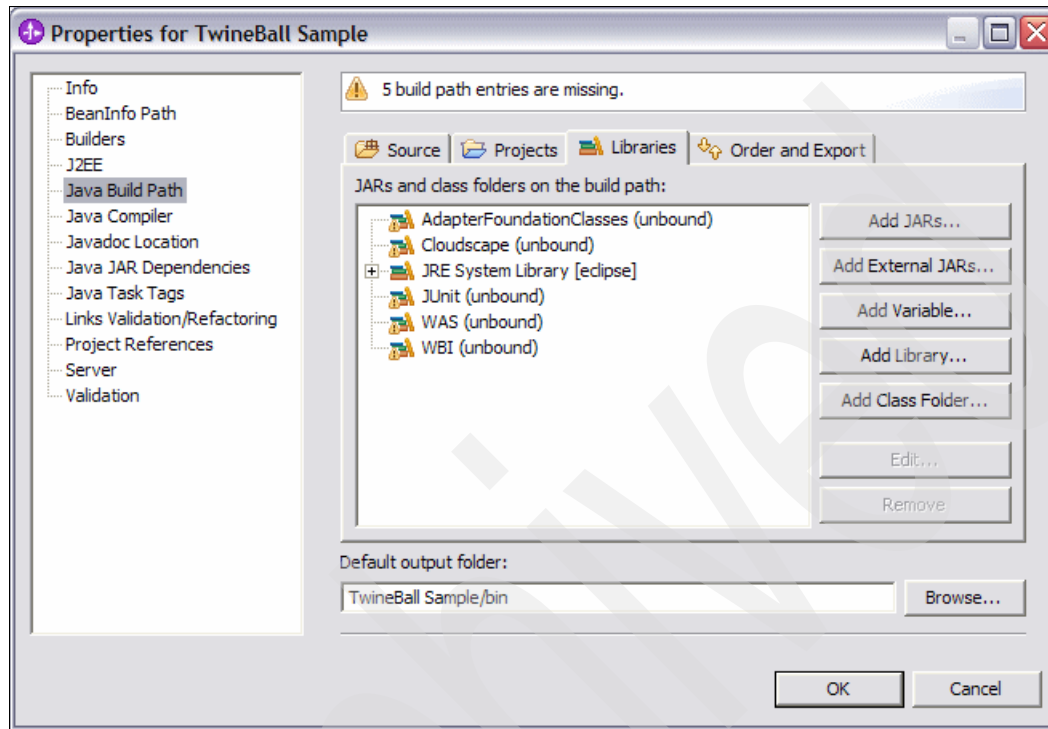


Figure 8-54 Properties for TwineBall Sample

- c. Notice that AdapterFoundationClasses, Cloudscape™, Junit, WAS, and WBI libraries are unbound. To bind these libraries, follow these steps:
 - i. Click **Window** → **Preferences** to open the Preferences dialog box ,then select **User Libraries** as shown in Figure 8-55.

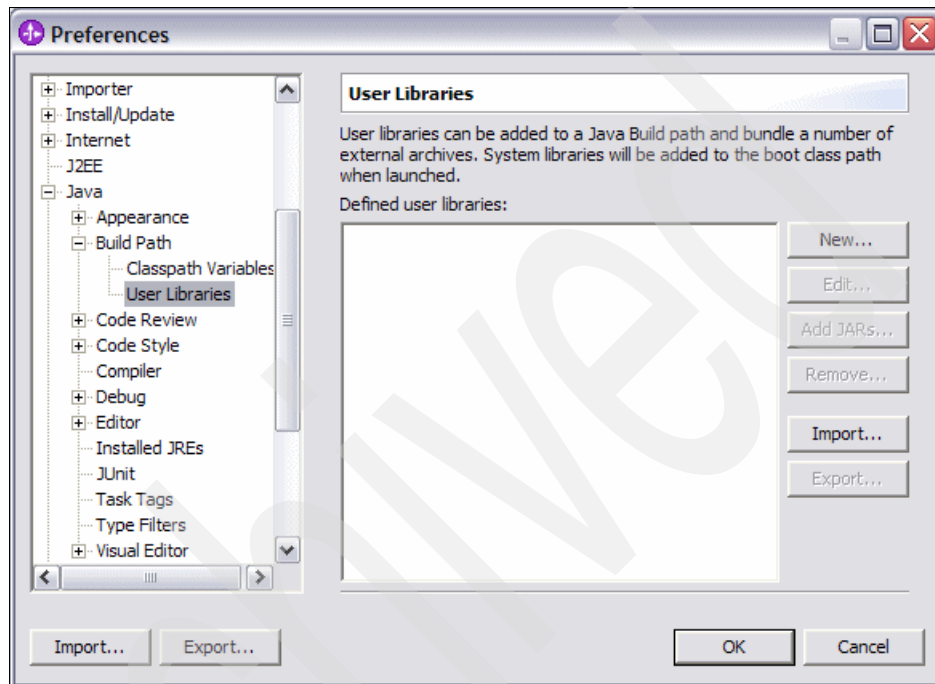


Figure 8-55 Preference dialog box

- ii. Click **New** to open a **New User Library** dialog box. Enter JUnit in the User library name text field then click **OK**.
- iii. In the Preferences dialog box, click **Add Jars** button to open a file chooser dialog box. Select **junit.jar** file from the following directory path to the new user library:


```
<WebSphere Integration Developer install location>\eclipse\plugins\org.junit_3.8.1\
```
- iv. Repeat step ii to create **AdapterFoundaionClasses** user library then repeat step iii to add **CWYBS_AdapterFoundation.jar** file from this directory path:

```
C:\Program Files\IBM\ResourceAdapters\AdapterToolkit\adapter\base\lib
```

- v. Repeat step ii to create **Cloudscape** user library then repeat step iii to add **db2j.jar** file from this directory path:

<WebSphere Integration Developer install location>\runtimes\bi_v6\cloudscape\lib

- vi. Repeat step ii to create **WBI** user library then repeat step iii to add all jar files from this directory path:

<WebSphere Integration Developer install location>\runtimes\bi_v6\lib

- vii. Close the **preference** dialog box then select **Project** → **Clean** to rebuild the project. When rebuild is done, you should not see any errors in the Problems view.

- viii. Browse the source code of TwineBall adapter.

Note: Sometimes, closing and reopening WebSphere Integration Developer can remove some of the errors and warnings in the problems view.

In this book, how TwineBall adapter works is not demonstrated. Instead, WebSphere Adapter Toolkit is used to create a custom adapter called RedMaintenance adapter. This adapter connects sample RedMaintenance EIS to WebSphere Process Server. For more information about TwineBall adapter see the *WebSphere Adapter Toolkit User Guide*.

The sample business scenario

This chapter describes a sample business scenario and a corresponding integration scenario that help automate the sample business. We then focus on a specific part of the scenario where a custom adapter is needed to facilitate Enterprise Information System (EIS) integration.

- ▶ Sample scenario overview
- ▶ Sample integration scenario overview
- ▶ RedMaintenance application overview

9.1 Sample scenario overview

To help you understand the context in which adapters are used, we created a sample business scenario. 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. This is 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 as Figure 9-1 shows.

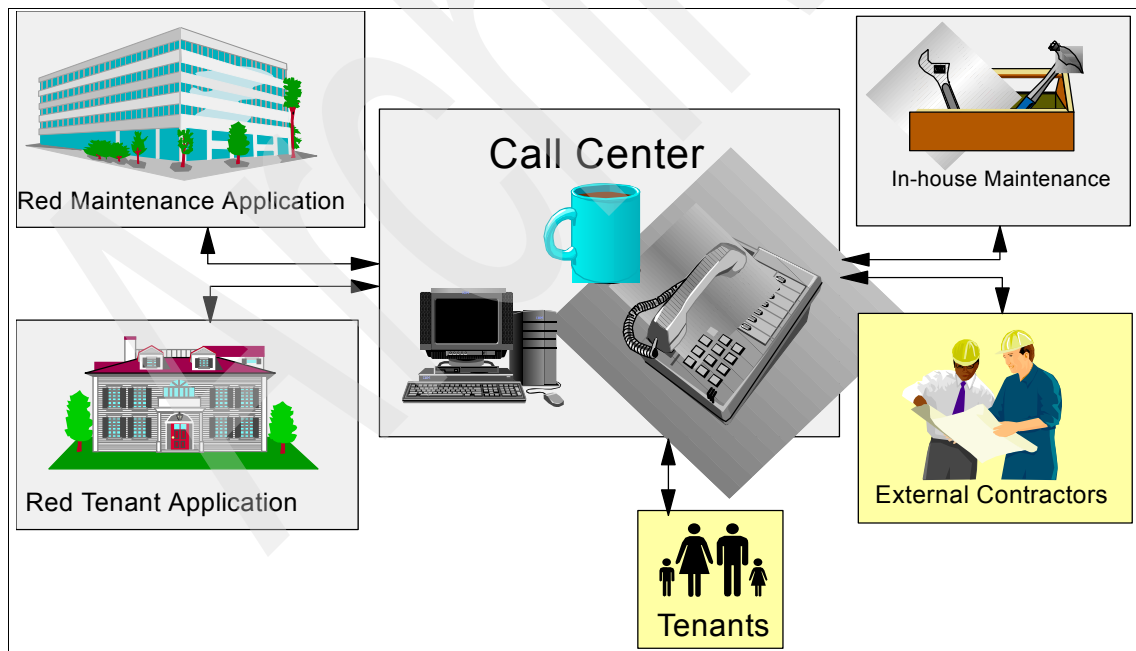


Figure 9-1 Current call center environment

The requirements

Management requires that the solution conform to the following conditions:

- ▶ 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 independently by a small software company.

9.2 Sample integration scenario overview

In our scenario, the two applications are integrated using a combination of adapters and the WebSphere Process Server:

- ▶ WebSphere Process Server provides a single, standards-based programming model to realize SOA. Service components represent a higher-level abstraction of the underlying IT and offer rich constructs to realize dynamic processes for business flexibility.
- ▶ Adapters allow you to quickly and easily create integrated processes that exchange information between Enterprise Information Systems through an integration broker. The adapters service-enable your applications by connecting them to WebSphere Process Server.

Figure 9-2 illustrates the new, integrated solution.

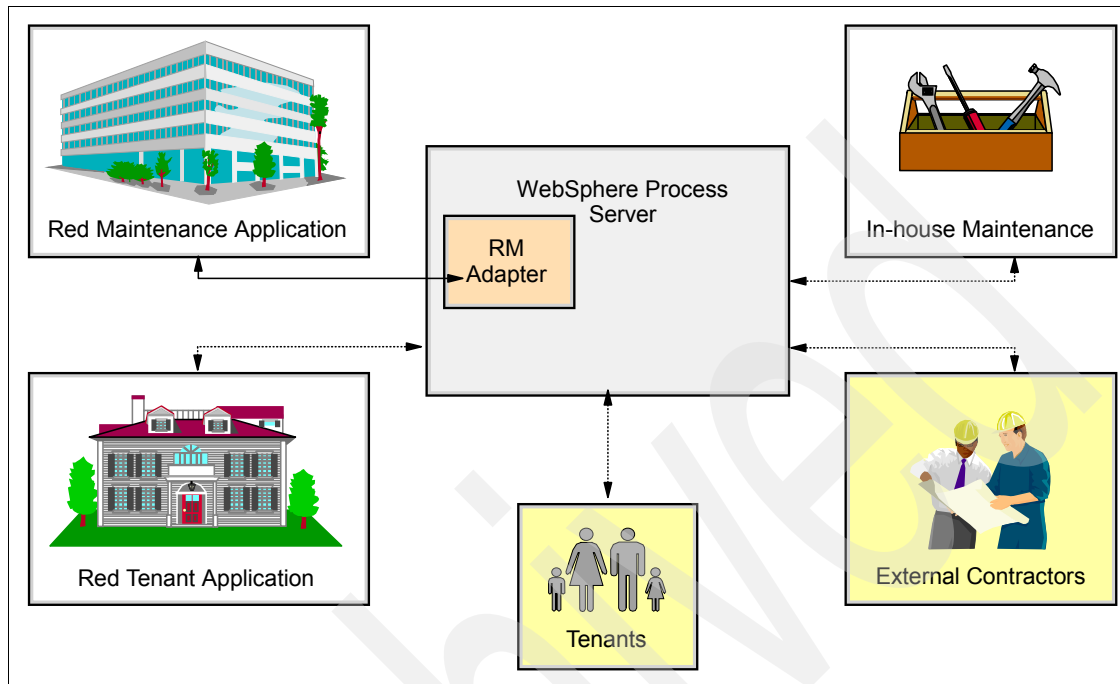


Figure 9-2 New integration with WebSphere Process Server and adapters

Discussion of a complete integration scenario using WebSphere Process Server can be complex and lengthy. That discussion is also outside the scope of this book because this book focuses on adapter development. It is assumed that the RedTenant application, external contractor integration, and Web interface for tenants has been out-sourced to an external company. It is further assumed that the only task remaining to complete this integration scenario is to develop a custom adapter for RedMaintenance application. This adapter allows bidirectional data communication between WebSphere Process Server and RedMaintenance.

9.3 RedMaintenance application overview

When you begin adapter development, it is essential to understand the environment and applications in that environment. The following sections describe:

- ▶ The RedMaintenance application environment.
- ▶ The entity relationship model used by RedMaintenance.

- ▶ Installation procedures for RedMaintenance.
- ▶ RedMaintenance APIs.

9.3.1 Application environment

These are some of the key points an adapter developer has to understand in order to develop a good adapter:

- ▶ Operating system
- ▶ Programming language
- ▶ Application architecture
- ▶ Database type
- ▶ Distributed application

9.3.2 Entity Relationship model

The Entity Relationship (E-R) model illustrates the internals of RedMaintenance application. Although it is not necessary for the adapter developer to know about it, the information provided is useful and facilitates the understanding of this simple application. The business entities of RedMaintenance are modeled as tables in a DB2® Database, as previously described.

Figure 9-3 shows the E-R of RedMaintenance application.

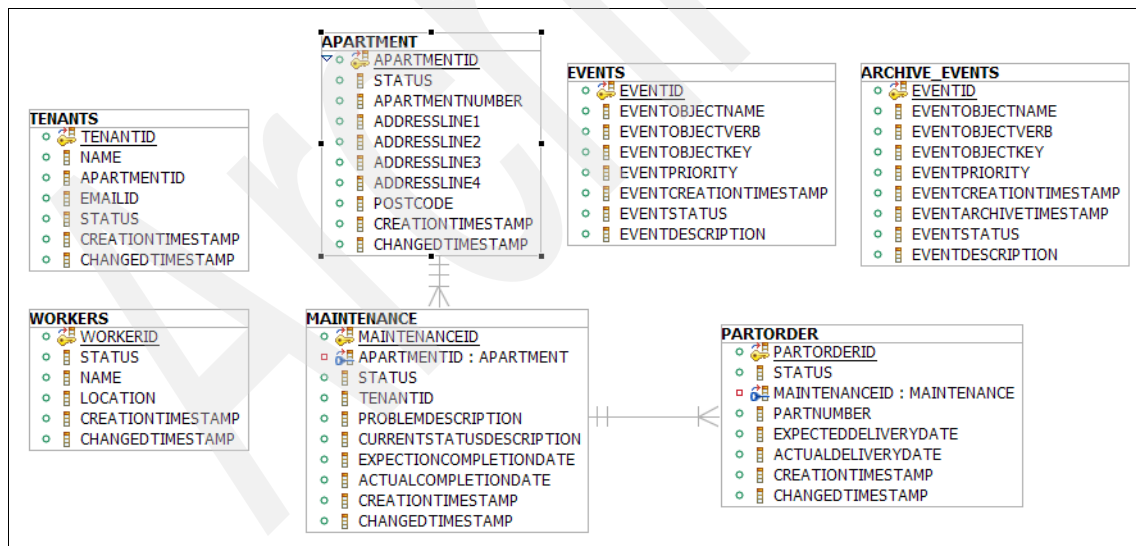


Figure 9-3 RedMaintenance E-R model

The entities are summarized as follows:

- ▶ WORKERS represents the maintenance workers.
- ▶ TENANTS represents the set of people who rents the different apartments managed by the company.
- ▶ MAINTENANCE describes the maintenance orders.
- ▶ PARTORDER represents a part of the maintenance order.
- ▶ EVENTS represents the storage of change notifications inside the another entities of the model.
- ▶ ARCHIVE_EVENTS represents is the storage of archived events.

This information shows that the business objects candidates are Worker, Tenant, Maintenance, and PartOrder.

Note: The business objects are referred commonly as *singular nouns*.

The columns of these entities are then associated with the attributes names of the respective business objects (preserving the type information).

9.3.3 Installing and setting up the RedMaintenance application

The first step before analyzing the RedMaintenance API is to ensure that the RedMaintenance application is correctly installed and configured.

This application uses a DB2 Universal Database as data repository, so it has to be installed before following these steps:

1. Download the application code and database scripts from the IBM Redbooks Web site. Look for the RedMaintenance application in:

<ftp://www.redbooks.ibm.com/redbooks/SG246387/SG246387.zip>

Unzip the file in an specific directory (for example, C:\SG246387).

2. Create and populate the database with the sample data:

- a. Open a DB2 command window.
- b. Navigate to the schemas folder of SG246387.
- c. Enter the following command:

```
CreateSG246387DB DB2 db2admin itso4you
```

(DB2 is the DB2 instance name. db2admin is the db2 administrator user id and itso4you is the db2administrator password).

- d. 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, you can use the PopulateSampleData.txt file located in the Schemas folder.
- e. In Figure 9-4, we imported the file as a script into the DB2 Command editor, and ran the script from there.

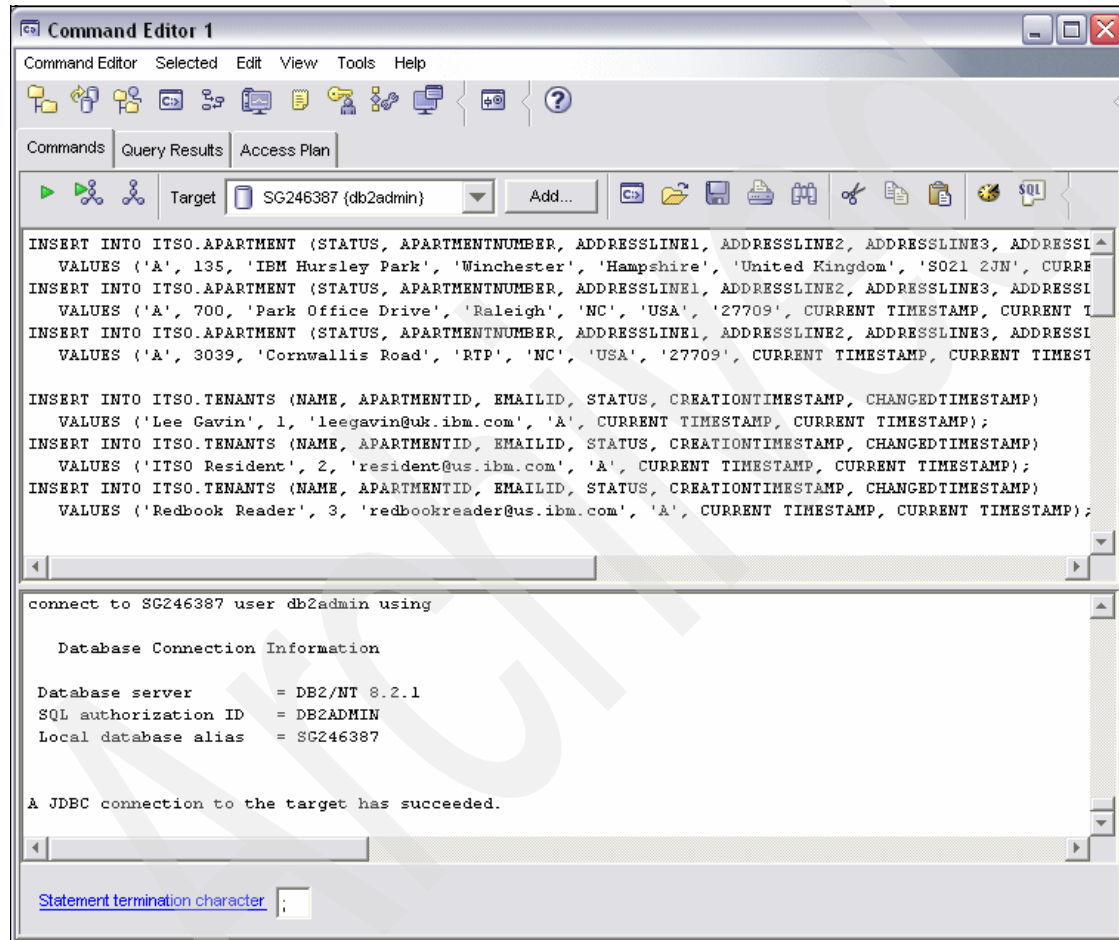


Figure 9-4 Import script in DB2 Command Center

- f. Using your DB2 tool, check that the database exists, with the correct tables and the sample data. Again, we used the DB2 Command Center. See Figure 9-5 on page 228.

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 9-5 RedMaintenance data sample

- To run the application, the next step is to ensure that the configuration file is set correctly. Using a text editor, open the file named **config** in the RedMaintenance folder. It is available for download from Appendix A, “Additional material” on page 435. Modify the database password to ensure that it is correct, then **save** and **close** this file. See Example 9-1.

Example 9-1 Description of config file

```
# Configuration properties for AppStart
databaseMaxConn=5
databaseURL=jdbc:db2:SG246387
databaseUserName=db2admin
databaseUserPwd=itso4you
jdbcDriverClass=COM.ibm.db2.jdbc.app.DB2Driver
```

- Start the **rmiregistry** command.

The **rmiregistry** command creates and starts a remote object registry on the specified port on the current host. If 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 Remote Method Invocation (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 provides application-specific support for finding other objects.

Open a command window and type **rmiregistry**.

Leave this window open in the background.

5. Create a shortcut on the desktop for starting the application engine, as shown in Figure 9-6. Ensure that the start directory is RedMaintenance.

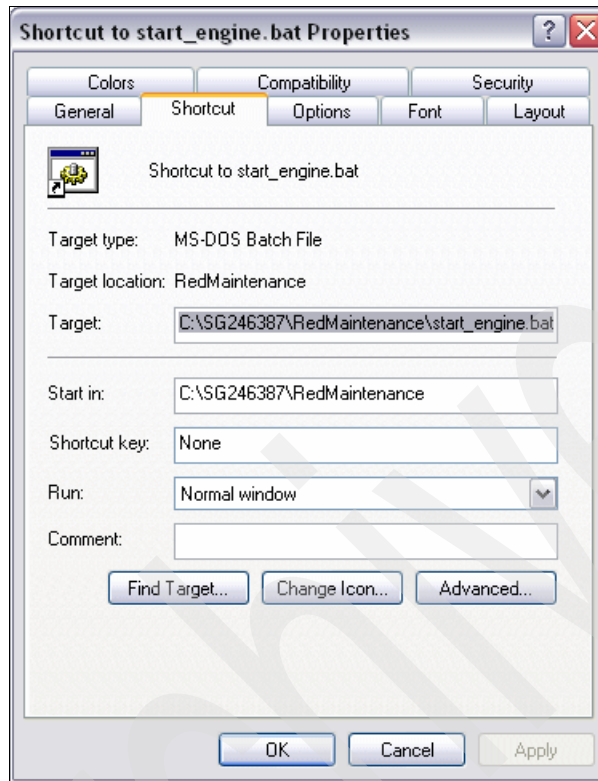


Figure 9-6 RedMaintenance Shortcut

6. Using this shortcut, start the application. In the command window, you can see the startup parameters used for the application. Leave this window open. See Example 9-2.

Example 9-2 Start window

```
C:\SG246387\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:SG246387
databaseUserPwd=itso4you
***** Driver Class is :COM.ibm.db2.jdbc.app.DB2Driver:
```

7. You also see a GUI window that is used for displaying RedMaintenance activity, for example creating a new apartment record, updating maintenance records, and so on. This useful for problem determination. See Figure 9-7.



Figure 9-7 GUI log

You now have the basic application components in place. You can move to the integration of these components and initial testing of the adapter environment.

9.3.4 RedMaintenance API

After the installation and running of RedMaintenance, the tasks needed to know to develop an adapter to this application are:

- ▶ Connecting to the application
- ▶ Accessing the business objects
- ▶ Accessing the event store
- ▶ Status management
- ▶ Inbound considerations

Connecting to the application

The connection to RedMaintenance is based on Java Remote Method Invocation (RMI). Example 9-3 shows the necessary code to connect to RedMaintenance.

Example 9-3 Testing RMI connection to RedMaintenance Application

```
package com.ibm.itso.sab511.tests.rmAppTest;

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;

import com.ibm.itso.rm.common.RMServerRMIInterface;

public class TestRMIConnection {

    public static void main(String[] args) {

        String host = "localhost";
        String appLocation = "rm";
        String name = "/" + host + "/" + appLocation;

        if (System.getSecurityManager() == null)
            System.setSecurityManager(new RMISecurityManager());

        RMServerRMIInterface eisInterface;
        try {
            eisInterface = (RMServerRMIInterface) Naming.lookup(name);
            // Execute some method of RMServerRMIInterface
            System.out.println("Connected");
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (RemoteException e) {
            e.printStackTrace();
        } catch (NotBoundException e) {
            e.printStackTrace();
        }
        eisInterface = null;
        System.out.println("Disconnected");
    }
}
```

Note: To run this simple application, the considerations of classpath configuration, security (policy file) and RMI codebase loading are needed to be set properly.

More information about RMI is available at the following link, *Java distributed objects: Using RMI and CORBA*:

<http://java.sun.com/developer/onlineTraining/Programming/BasicJava1/rmi.html>

Accessing the business objects

RedMaintenance is especially designed to provide a unique way of invocation and access to all of its application-specific business objects.

The following steps are needed to correctly access or modify an application business object:

1. Implement the RMI connection coding (refer to the Example 9-3 on page 231).
2. Create an instance of the RMDDataImplementation class, using the default constructor.
3. Set the component name of the data. The allowed values are: Apartment, Maintenance, PartOrder, Tenant, and Worker.
4. Set the execution type (verb) requested for that object. The allowed values are:
 - Create
 - Retrieve
 - Update
 - Delete
5. If the operation is not retrieving, all the necessary attributes must be set. However, if operation is retrieved then, only the primary key is needed.
6. Execute the operation in the RMServerRMIInterface instance.
7. Control the execution of this operation.

The Example 9-4 describes the steps defined above in a code segment.

Example 9-4 Code used to invoke an application business object or RedMaintenance

```
// ... Connection has been created

RMDDataImplementation request = new RMDDataImplementation();
request.setComponentName("Apartment");
request.setExecutionType("Create");
request.setAttr("ApartmentNumber", "101");
request.setAttr("AddressLine1", "101 RedStreet");
request.setAttr("AddressLine2", "RedCity, NC");
request.setAttr("AddressLine3", "");
request.setAttr("AddressLine4", "");
request.setAttr("PostCode", "12345");
```


Because RMDDataImplementation encapsulates the application business objects information, the knowledge of the available attributes can be obtained from the E-R model and from the classes implementation. Figure 9-9 shows the application business objects used by RedMaintenance and the methods they implement.



Figure 9-9 RedMaintenance application business objects

The RedMaintenance application supports the Create, Update, Retrieve and Delete operations for all the application business objects described above.

Accessing the event store

RedMaintenance manages the event store through two entities in its database: EVENTS and ARCHIVE_EVENTS, but from the perspective of an adapter developer, the events' operations are invoked through the RMDDataImplementation class. In other words, RedMaintenance uses the same approach to manipulate application business objects and events.

The difference in this case is in the structure of the Event entity and in the fact that the Event permits an additional execution type of invocation, called Archive. This additional execution type permits the archive process of events in the ARCHIVE_EVENTS table.

Note: In this version of WebSphere Adapter, the management of archive events has changed. Now the responsibility of archiving events is delegated to the adapter, so the EIS archive repository is not used.

The steps necessary to manipulate an event are very similar to any of the business objects described before. The considerations perform that implementation are:

- ▶ The component name is Event.
- ▶ The attributes of an event are shown in the Figure 9-10.



Figure 9-10 Methods of the Event class

Status management in outbound processing

In order to invoke RedMaintenance operations, the adapter developer has to know that there are special requirements applicable for Update operations on the application-specific business objects.

These requirements are related with the use of the status attribute. For every application business object, there exists a status attribute. In each of these business objects its value has to match one of the allowed values.

Table 9-1 shows the list of allowed values.

Table 9-1 Allowed values for status attribute in the RedMaintenance ASBOs

Application Business Object	Allowed status attribute values
Apartment	A (Active) D (Deleted) I (inactive)
Maintenance	A (Active) C (Completed) D (Deleted) P (In progress) I (Internal contractor) X (External contractor)
PartOrder	N (New) O (Ordered) R (Required) D (Deleted)
Tenant	A (Active) D (Deleted)
Worker	A (Active) D (Deleted)

The adapter developer must ensure that before invoking any Update operation, the application business object must have its corresponding status attribute with the value of A (Available). Otherwise, the Update operation fails.

Inbound considerations

The event triggering mechanism is dispatched in RedMaintenance only for the Create and Update operations upon the Maintenance object. The implementation of the events' generation for the other application-specific business objects is not available. In order to provide it, triggers or modifications of the original source code are needed. However, these tasks are out of the scope of this redbook and they could be implemented as a further exercise.

Adapter design and specification

This chapter discusses the design and specifications for a custom adapter that can connect to the RedMaintenance application. This is called the RedMaintenance adapter. The following topics are discussed:

- ▶ Adapter design technical assessment
- ▶ RedMaintenance adapter design and specification

10.1 Adapter design technical assessment

The first step in adapter development is to determine if you truly need to build an adapter. Perhaps using a technology adapter such as IBM WebSphere Adapter for Java Database Connectivity (JDBC) is a better approach. Whether to build a custom adapter or not is not an easy decision. While 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 technically possible to do so. For example, does your application have an application programming interface (API) that the adapter can use 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 can dictate 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, the decision to build a custom adapter is dependant on your requirement. The information provided in this book aims to help make the decision process a bit easier.

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

Understanding the RedMaintenance application

When you begin adapter development, it is crucial to understand the environment and applications in that environment. In Chapter 9, “The sample business scenario” on page 221, we provide an overview of the RedMaintenance application. We discuss the RedMaintenance’s data model, installation, and application programming interfaces (APIs). In the following sections, we further explore these topics.

Examining the application environment

As mentioned in Chapter 9, “The sample business scenario” on page 221, you can use a list of topics and questions to obtain an understanding of the aspects of an application that affect adapter development. Some of the key points about our major application are:

- ▶ Operating system:
 - Supports Java 1.4 or above.
 - Runs on a Microsoft Windows platform.
- ▶ Programming language
Application written in Java
- ▶ Application execution architecture
Multi-threaded application using an Remote Method Invocation (RMI) communications mechanism for connectivity
- ▶ Database Type:
 - IBM DB2 DB2 Universal Database™ (UDB)
 - Relational
- ▶ Distributed application:
 - Is the application distributed across multiple servers?
Currently executes on a single platform (Windows).
 - Is the application database distributed across multiple servers?
Currently executes on a single platform.

Early in the project planning phase, you must determine what role the adapter must perform for the application:

- ▶ Request processing (outbound processing)
Update application data at the request of an integration broker
- ▶ Event notification (inbound processing)
Detect application events and send notification of events to the integration broker.

To be bidirectional, the adapter needs to support both event notification and request processing. Our application has this capability.

Examining the RedMaintenance API

RedMaintenance application provides an API that includes all of the following features:

- ▶ Support for object-level Create, Retrieve, Update, and Delete (CRUD) operations

- ▶ Encapsulation of all of the application business logic
- ▶ Support for delta and after-image operations

The recommended approach to adapter development is to use the API that the application provides. The use of an API helps ensure that adapter interactions with applications 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 Structured Query Language (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 adapter does not bypass application business logic. This aspect of the application has a major impact on adapter design because it affects the amount of coding that the adapter requires. The easiest application for adapter 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 adapter probably requires more coding.

The following questions asked about the RedMaintenance API can help you decide if it is a good candidate for an adapter:

- ▶ Is there an existing mechanism that the adapter can use to communicate with RedMaintenance application?

Yes.

- ▶ Does the RedMaintenance 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.

- ▶ Does the RedMaintenance API provide access to all attributes of a data entity?

No, some data entities are internal to the application.

- ▶ Are there inconsistencies in the RedMaintenance API implementation?

No.

- ▶ Is the navigation to the CRUD functions the same regardless of the entity?
Yes.
- ▶ Describe the transaction behavior of the RedMaintenance API.
Each Create, Update, or Delete operation operates within its own transactional unit.
- ▶ Does the RedMaintenance API allow access to the application for event detection?
Yes.
- ▶ Is the RedMaintenance API suited for metadata design? APIs that are forms, table, or object-based are good candidates.
Yes. All RedMaintenance APIs are object-based.
- ▶ Does the RedMaintenance API enforce application business rules?
Yes. The RedMaintenance API interacts at object level.

Assessment result

The RedMaintenance application is an excellent candidate for a metadata-driven adapter. By using a series of adapters and a WebSphere Process Server for all of your connected applications, you are enabling more business process automation and synchronizing of data. The solution can also be expanded in the future.

10.2 RedMaintenance adapter design and specification

The RedMaintenance resource adapter enables bidirectional connectivity for integration between WebSphere Process Server and RedMaintenance application. RedMaintenance resource adapter supports processing of requests (outbound processing) from application components running on WebSphere Process Server. It also supports the processing of events (inbound processing) occur in RedMaintenance and deliver these events to predefined endpoints in WebSphere Process Server.

10.2.1 Architecture overview

The RedMaintenance resource adapter enables any Service Component Architecture (SCA) component running on WebSphere Process Server to connect to RedMaintenance and interact with it. It also has the capability to gather RedMaintenance's metadata by providing enterprise metadata discovery support. The following are outbound, inbound, and enterprise metadata discovery scenarios for RedMaintenance adapter.

Outbound scenario

SCA components passes a business object to the adapter when invoking one of the adapter operations. The adapter processes incoming business object. It does this by invoking the appropriate RedMaintenance API to Create, Retrieve, Update, Delete records in RedMaintenance. The result of the changes to RedMaintenance is return to the calling application component in a business object.

Inbound scenario

When a RedMaintenance entity object is created, updated or deleted, an event is generated and recorded in an event store or table. The adapter continuously polls RedMaintenance's event store. If there is an event in the event store, the entity object processes it. In each event processing cycle, the adapter checks the event store, retrieves the corresponding record from RedMaintenance, builds a business object and sends it to a message endpoint. SCA application component is the message endpoint.

Enterprise metadata discovery

In order to use the adapter for inbound and outbound operations on WebSphere Process Server, we need to define RedMaintenance business objects, create inbound outbound interface, and create SCA EIS import export binding. Creating these artifacts manually can be quite tedious and time consuming.

RedMaintenance adapter implements the contracts specified in the Enterprise Meta Discovery (EMD) specification. This allows WebSphere Integration Developer tool to do the following through RedMaintenance's implementation of EMD interfaces:

1. Connect to RedMaintenance.
2. Mine RedMaintenance's proprietary metatdata, services, and business objects.
3. Browse the discovered RedMaintenance's services and business objects tree.
4. Select the nodes you want.
5. Create service description for the selected nodes.

Instead of coding our adapter from scratch, we leverage the generically implemented business logic in WebSphere Adapter Foundation Classes to create RedMaintenance adapter.

Figure 10-1 shows RedMaintenance adapter is consist of adapter foundation classes and RedMaintenance specific subclasses that extends foundation classes.

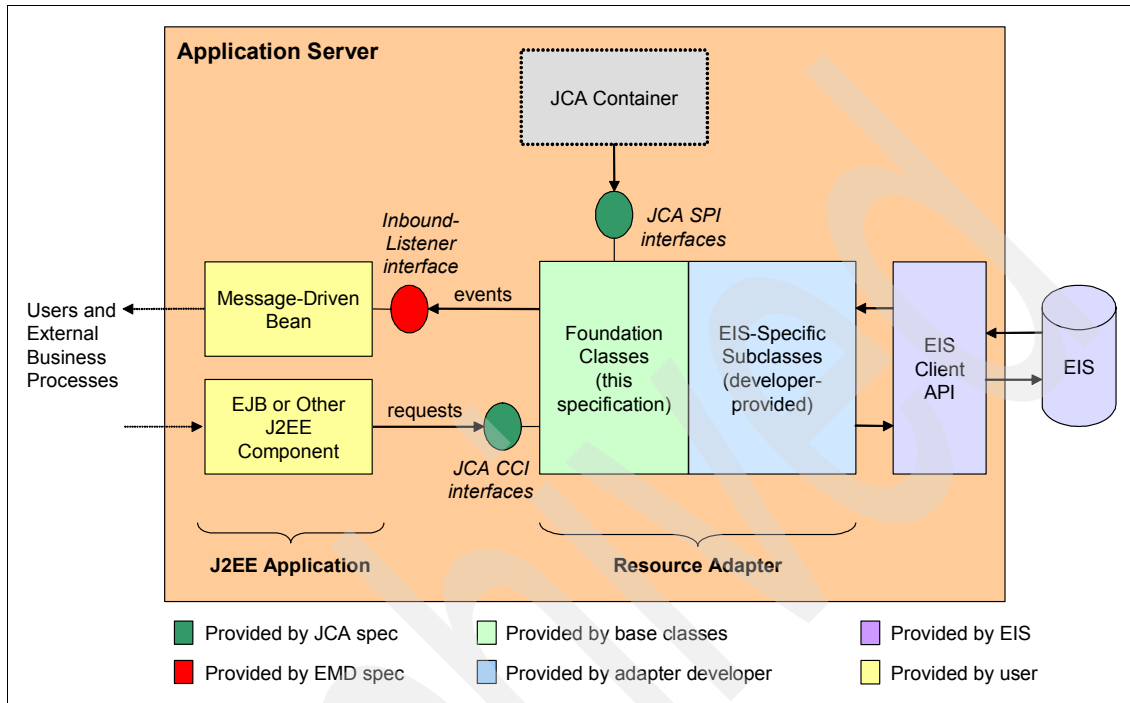


Figure 10-1 RedMaintenance resource adapter architecture

10.2.2 WebSphere adapter foundation classes

The foundation classes include generic contracts and methods to help develop a working resource adapter. This saves the adapter developer from recoding common business logic when developing adapters.

[illegible]

General steps to build an adapter from foundation classes

1. Investigate and decide on configuration properties that need to be defined in the resource adapter:
 - a. Identify connection configuration properties to the Enterprise Information System (EIS). For example, host name, and port.
 - b. Identify outbound specific configuration properties.
 - c. Identify inbound event processing configuration properties. This can be a combination of properties in step a and step b.

- d. Identify any remaining configuration properties not specific to inbound or outbound.
2. Extend `WBIResourceAdapter` class and add properties identified in step d above to the deployment descriptor.

Note: When you add these new properties to the deployment descriptor, WebSphere Adapter Toolkit automatically creates the corresponding getter and setter methods in the sub-class of `WBIResourceAdapter`.

3. If the resource adapter supports inbound event processing via the event manager Quality of Service (QoS) described later in this document, modify your `WBIResourceAdapter` subclass to implement interface `WBIPollableResourceAdapter` and provide an implementation of interface `EventStore`. If you defined additional inbound properties in step 1 on page 244c on page 244 beyond what is already defined for class `WBIActivationSpec`, extend `WBIActivationSpec` and update your deployment descriptor to reflect this new subclass class under the supported activation specs.
4. Extend class `WBIManagedConnectionFactory` and add properties defined in step 1 on page 244a on page 244 to the RA deployment descriptor.

Note: When you add these new properties to the deployment descriptor, WebSphere Adapter Toolkit automatically create the corresponding getter and setter methods in the sub-class of `WBIManagedConnectionFactory`.

5. Extend `WBIManagedConnection` class and `WBIManagedInteraction` class. In your `WBIManagedInteraction` subclass provide logic for processing requests (that is, CRUD operations) while in your `WBIManagedConnection` simply provide the ability to generate a new `WBIManagedInteraction` instance.
6. Extend `WBIManagedConnection` class and provide logic to physically connect and disconnect to the EIS.
7. If you defined additional connection-specific properties in 1 on page 244 and b on page 244 beyond what is already defined in classes `WBIManagedConnectionRequestInfo` and `WBIManagedConnectionRequest` extend both and add these properties.

10.2.3 RedMaintenance adapter features

This section lists the RedMaintenance adapter features.

Runtime support

RedMaintenance adapter supports outbound and inbound in a managed environment only. This means it can only run in the context of an application server.

Application sign-on

Typically connecting to an EIS requires some type of authentication. In a J2EE Connector Architecture (JCA) environment, application authentication is referred to as *sign-on*. RedMaintenance is an open system, therefore it does not require any authentication for clients to connect and use its APIs.

Connection management

RedMaintenance adapter implements the adapter side of the JCA connection management contract. This allows the JCA compliant application server to manage the connection pooling to RedMaintenance.

Outbound support

The following operations are supported for outbound:

► Create

Create performs a recursive traversal of the incoming hierarchical business object. Creating records in RedMaintenance corresponding to each table as it traverses the incoming business object:

- Inserts the top-level business object in RedMaintenance
- Inserts each of its multiple-cardinality child business object in RedMaintenance

► Retrieve

Given a business object with a valid key attribute, the adapter performs a recursive traversal of the RedMaintenance database to retrieve the current object and all children objects:

- Retrieve the top-level business objects (BO) from RedMaintenance.
- Recursively retrieve all multiple-cardinality child business objects.

► Update

The adapter compares the incoming business object with the business object that is retrieved from RedMaintenance using the primary key specified in the top level business object of the incoming business object.

The adapter processes the update as follows:

- a. Uses the primary key of the incoming business object to retrieve the corresponding entity hierarchy from RedMaintenance. The adapter compares the retrieved business object with the incoming business object to determine if children business object requires change in RedMaintenance.
 - b. Updates all simple attribute of the retrieved business object
 - c. Processes each multiple cardinality child of the retrieved business object in one of the following ways:
 - If the child exists in both the incoming business object hierarchy and the retrieved business object hierarchy, the adapter recursively updates RedMaintenance.
 - If the child exists in the incoming business object hierarchy but not in the retrieved business object hierarchy, the adapter recursively creates it in RedMaintenance.
 - If the child exists in the retrieved business object hierarchy but not in the incoming business object hierarchy, the adapter recursively deletes it from RedMaintenance.
- Delete
- Given a business object with a populated key attribute, the adapter performs a recursive traversal of the RedMaintenance database to delete the current object and all children objects:
- Delete the top level BO from RedMaintenance.
 - Recursively Delete all multiple-cardinality child business objects.
 - Return a business object hierarchy tree of the deleted RedMaintenance entities. From this tree, the calling application component can obtain information about deleted objects.

Note: RedMaintenance adapter leverages the Command Pattern of WebSphere Foundation Classes to process these CRUD operations. See 4.11, “Command Pattern” on page 95 for Command Pattern details.

Transaction management

Transaction management allows the application server to take responsibility of managing transactions across multiple resource managers. Under JCA specification there are two types of transactions:

- ▶ Local transaction

A local transaction is managed internally by the resource manager. RedMaintenance does not support local transactions.

- ▶ XA transaction

An XA transaction requires coordination by an external transaction manager to manage transactions that span multiple resource managers. RedMaintenance does not support XA transactions either.

Inbound support

Inbound operations are supported through the use of Event Store. When data is changed in the RedMaintenance application tables in the database, the appropriate event (Create, Update, or Delete) is inserted into the event store along with relevant information such as key values. This is done by placing triggers on the respective tables. The RedMaintenance adapter polls the event store and retrieves a batch of events. These events are processed and each one is used to construct a business graph for the corresponding RedMaintenance object. The constructed business graph is then dispatched to the end points that have a subscription for the specific business object. Only After Image Support is provided for Inbound.

The following inbound operations are supported by RedMaintenance adapter:

- ▶ Create
- ▶ Update

Enterprise Service Discovery support

RedMaintenance adapter implements adapter side of the EMD contract. RedMaintenance EMD mines the business object, services, and metadata from RedMaintenance application in two ways.

- ▶ It access RedMaintenance's Object Discovery Agent (ODA) API to discover metadata.

Note: RedMaintenace is a sample application. Its ODA API does not actually query RedMaintenance's database. It returns a fixed data structure of metadata when invoked.

- ▶ It access RedMaintenance's database to discover metadata.

Typically, you only need to implement one of the listed methods for metadata discovery. In our example RedMaintenance EMD implementation, we demonstrate this can be done in two ways.

Archived

Implementing outbound request processing

This chapter explains how we implement outbound request processing for the RedMaintenance adapter. We provide details on how to use WebSphere Adapter Foundation Classes library to speed up adapter outbound processing development.

Creating a custom resource adapter that supports outbound request processing involves the following steps:

- ▶ Start the development project.
- ▶ Use WebSphere Adapter Toolkit to create adapter outbound stub classes.
- ▶ Implement stub classes to connect to Enterprise Information System (EIS).
- ▶ Implement stub classes to support various outbound operations.
- ▶ Create temporary Service Component Architecture (SCA) artifacts to test outbound operations during development.

11.1 Start the development project

From the previous chapters, we acquire essential information about RedMaintenance application data entities, application programming interfaces (APIs) and event management. These findings become the basis for a high-level design for RedMaintenance adapter. We now continue with the next phase of RedMaintenance adapter. Typically an adapter has three main components:

- ▶ Outbound request processing
- ▶ Inbound request processing
- ▶ Enterprise Metadata Discovery

Depending on the need of the adapter that you are developing, you can implement one or more of these adapter components. In this chapter, we discuss the outbound processing component of our sample RedMaintenance adapter and show you the steps to implement it.

As part of the adapter development process, we use WebSphere Adapter Toolkit to help generate the stub classes that are needed to satisfy the resource adapter side of the J2EE Connector Architecture (JCA) specification. We then implement these stub classes. In addition, the overall process of developing an adapter includes other tasks, such as developing application-specific business objects and generating SCA artifacts to bind the adapter into service component architecture of WebSphere Process Server. A quick checklist or overview of the tasks in the adapter development process includes the following:

1. Identify the application entities that the adapter makes available to other applications, and investigate the integration features that are provided by the application.
2. Use WebSphere Adapter Toolkit to create the stub classes that are required to be implemented to satisfy the adapter side of the JCA contracts. WebSphere Adapter Toolkit can also generate stub classes for Enterprise Metadata Discovery.
3. Identify adapter specific properties. For example, user name and password to connect to the application. Enter these properties in the adapter deployment descriptor.
4. Implement the stub classes to connect to the EIS.
5. Implement the stub classes to support various outbound operation. When you implement the outbound operations, you can leverage the services offered by the *Command Pattern*. Command Pattern is part of WebSphere Adapter Foundation Classes. It has generic business logic to help process outbound business objects.
6. Implement error and message handling for all adapter methods.

7. Build the connector.
8. Create SCA artifacts so you can test the adapter on WebSphere Process Server.
9. Test and debug the adapter on WebSphere Process Server, recoding as necessary.

11.2 Identify application properties and operations

In order for the adapter to connect to the RedMaintenance application, we need to identify the connection mechanism and connection properties. Once we have a connection to the RedMaintenance application, we need to identify the operations that a application running on WebSphere Process Server can invoke on the adapter. Adapter operations generally require sending application-specific business objects and receiving application-specific business objects. These application-specific business objects must be identified when we develop our custom resource adapter.

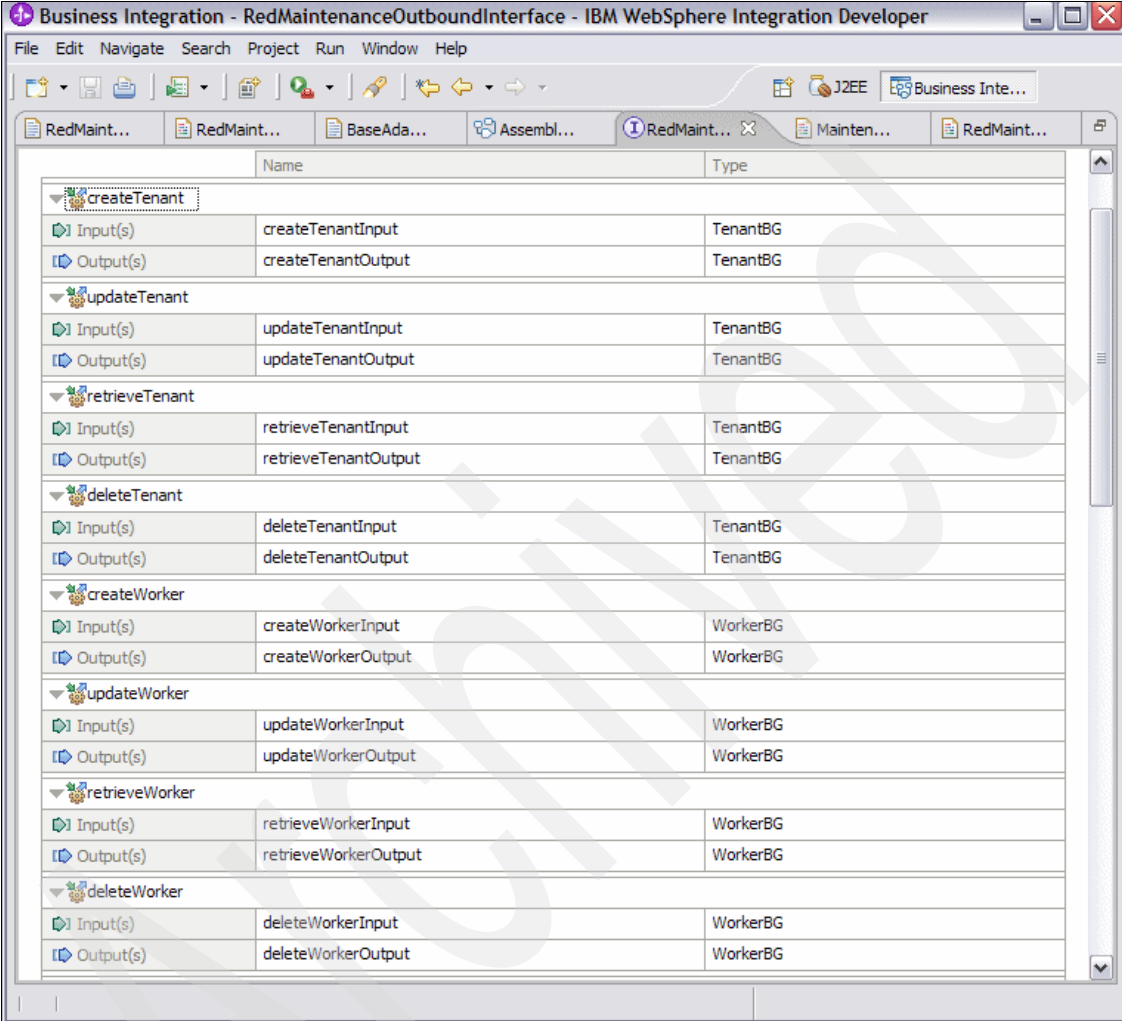
11.2.1 Connection properties

RedMaintenance is a Java application. It uses Java RMI to bind its external interface to a Java RMI registry. The client performs a lookup to get a remote reference to RedMaintenance's external interface. The client use this reference, to invoke RedMaintenance's APIs. The connection properties required by our custom adapter are the RedMaintenance's server URL and the Remote Method Invocation (RMI) name it uses to bind its external interface.

11.2.2 Outbound operations

From the integration requirement, our adapter must support data synchronization between RedTenant and RedMaintenance. The requirement also specifies that the tenant should be able to access Maintenance records over the Web. To support these requirements, our custom adapter must support Create, Retrieve, Update, Delete (CRUD) operations on all of RedMaintenance's entities. The list of RedMaintenance entities is in Chapter 8, "Setting up the development environment" on page 159.

See examples of these operations for Tenant and Worker entities in Figure 11-1.



The screenshot displays the IBM WebSphere Integration Developer interface. The title bar reads "Business Integration - RedMaintenanceOutboundInterface - IBM WebSphere Integration Developer". The menu bar includes File, Edit, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations and development tools. The main workspace shows a project explorer on the left with folders for RedMaint..., BaseAda..., Assembl..., and RedMaint... The central area displays a table of operations for two entities: Tenant and Worker. Each entity has four operations: create, update, retrieve, and delete. Each operation is represented by a tree view with an icon and a table of input and output messages. The input and output messages are listed in the table below.

Name	Type
createTenant	
Input(s)	createTenantInput
Output(s)	createTenantOutput
updateTenant	
Input(s)	updateTenantInput
Output(s)	updateTenantOutput
retrieveTenant	
Input(s)	retrieveTenantInput
Output(s)	retrieveTenantOutput
deleteTenant	
Input(s)	deleteTenantInput
Output(s)	deleteTenantOutput
createWorker	
Input(s)	createWorkerInput
Output(s)	createWorkerOutput
updateWorker	
Input(s)	updateWorkerInput
Output(s)	updateWorkerOutput
retrieveWorker	
Input(s)	retrieveWorkerInput
Output(s)	retrieveWorkerOutput
deleteWorker	
Input(s)	deleteWorkerInput
Output(s)	deleteWorkerOutput

Figure 11-1 Examples of CRUD operations on Tenant and Worker entities

11.3 Generate outbound stub classes

The IBM WebSphere Adapter Toolkit contains everything you need to create a resource adapter.

To implement outbound support for our custom adapter, we use the toolkit:

1. Launch the New J2C Resource Adapter Project Wizard to create a new resource adapter project and generate adapter stub classes:

Note: These are the same steps as in 8.4, “Create a Hello World adapter” on page 181.

- a. Start **WebSphere Integration Developer**.
- b. Select a new workspace when prompted.
- c. WebSphere Integration Developer’s integrated development environment appears. Close the **Welcome view** to see the business integration perspective.
- d. From the menu, select **File** → **New** → **Project**.
- e. In the Select a wizard dialog box, select **Adapter Toolkit** → **J2C Resource Adapter Project**.
- f. Click **Next**.
- g. Enter **RedMaintenance** as the adapter project name in the Connector Project dialog.
- h. Click **Next**.
- i. In the J2C Resource Adapter Properties dialog, enter the following information in the Adapter Name, Package Name, Class Name Prefix fields:
 - RedMaintenance
 - com.ibm.itso.sab511
 - RM
- j. Click **Next**.
- k. In the Generation Options dialog box, select **IBM WebSphere Resource Adapter**.

Note: We chose IBM WebSphere Resource Adapter type to leverage the services provided by WebSphere Adapter Foundation Classes. The foundation classes helped us to speed up development for adapters that are targeted for WebSphere Process Server. For differences between IBM WebSphere adapter and plain JCA resource adapter see Chapter 2, “Adapter basics” on page 33.

- l. In the same dialog box, select **Generate Outbound Adapter Classes** and **Generate Command Pattern Classes**.

Note: If your adapter supports inbound operation and Enterprise Metadata Discovery, you can select them here. If your adapter supports bidirectional languages, for example Hebrew and Arabic, you can select **Generate Bidi Support classes**.

- m. Click **Finish** to start generating selected components.
 - n. Click **Yes** if you are prompted to switch your perspective to J2EE.
2. Using the toolkit's Resource Adapter Deployment Descriptor editor, add new properties and set security permissions for the adapter.

As we mention in the previous section, we need to know the RedMaintenance server link as well as RMI name of RedMaintenance's external interface. Instead of hard coding these values, we can create properties for them in the toolkit's Resource Adapter Deployment Descriptor editor. In Chapter 2, “Adapter basics” on page 33, we explain that resource adapter adapters run inside WebSphere Process Server. In order for our custom adapter to connect to RedMaintenance server, we must grant the appropriate security permissions to allow Java RMI. Security permissions can be set in the toolkit's Resource Adapter Deployment Descriptor editor.

The following steps show you how to add new properties and set security permissions in the deployment descriptor editor:

- a. Open Resource Adapter Deployment Descriptor editor as shown in Figure 11-2.

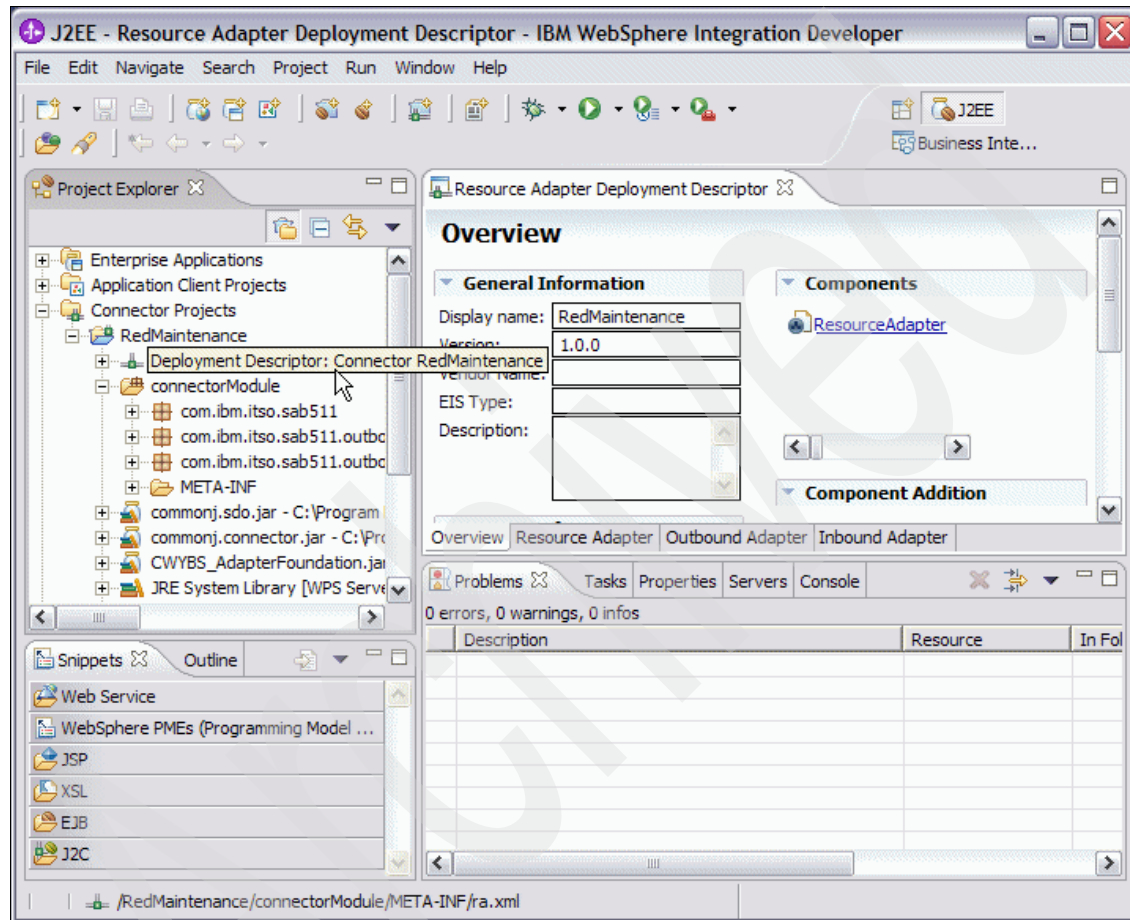


Figure 11-2 Adapter Deployment Descriptor editor

- b. In the Resource Adapter Deployment Descriptor editor, select **Outbound Adapter**.
- c. Under Connection Definitions, select **javax.resource.cci.ConnectionFactory**.

d. Click **Add** as shown in Figure 11-3.

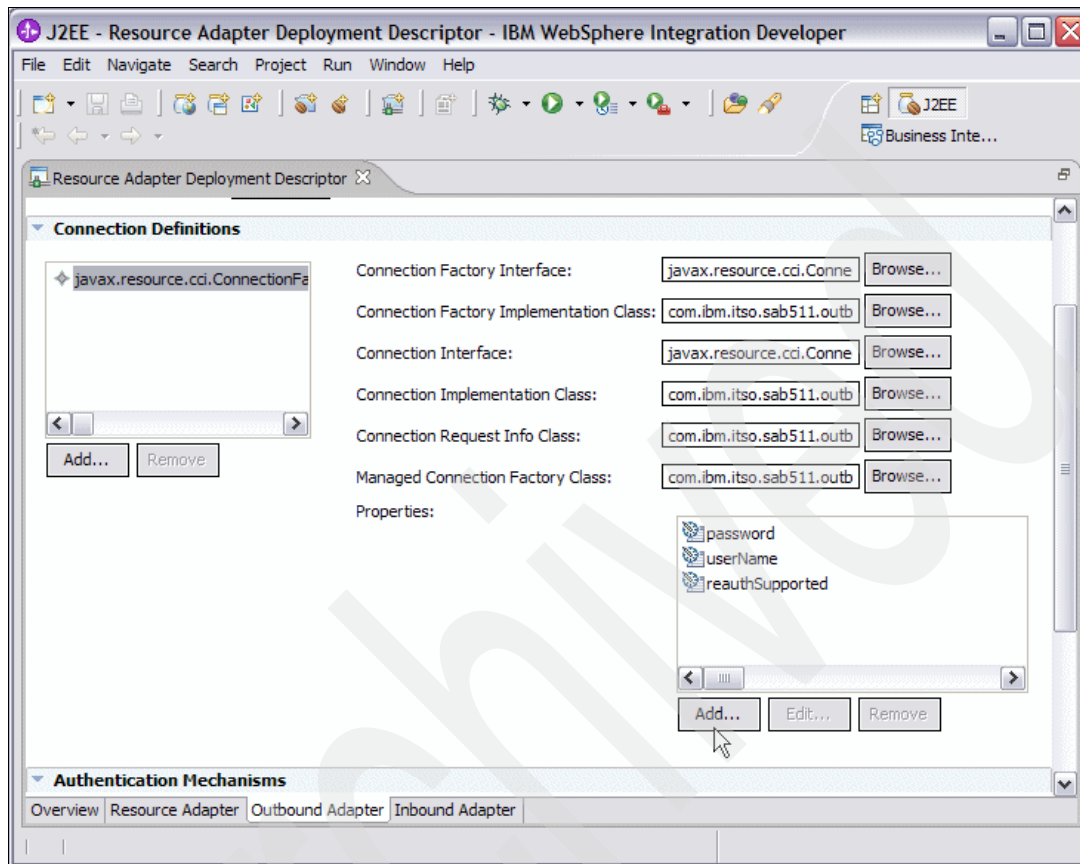


Figure 11-3 Add outbound properties

- e. On the Add Config Property dialog box enter the values as show in Figure 11-4.

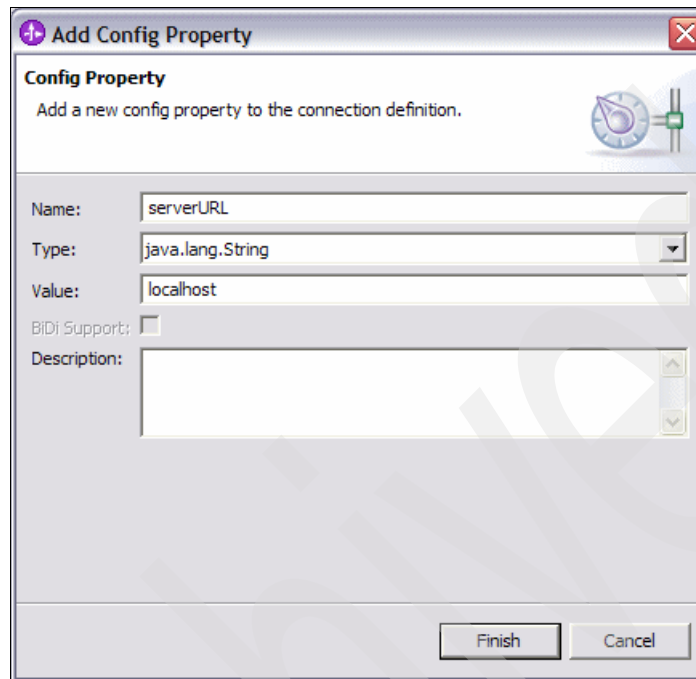


Figure 11-4 Add server URL property

- f. Click **Finish**.
- g. Repeat steps d and e to add property `rmiName`, type `java.lang.String` and value `rm`.
- h. Click **File** → **Save**.

Note: WebSphere Adapter Toolkit automatically creates the fields `serverURL`, `rmiName` along with their getter and setter methods in `RMManagedConnectionFactory.java` and `RMConnectionRequestInfo.java` files. We can access these values when implementing adapter connection to RedMaintenance.

Typically, an EIS client must be authenticated when connecting to EIS. However, RedMaintenance does not require any authentication for connecting to it and for client to access to its APIs.

- i. Select **existing reauthSupported** property.

- j. Click **Edit**.
- k. Set the value of this property to `false`.
- l. Click **Finish**.
- m. In Resource Adapter Deployment Descriptor editor, select **Resource Adapter**.
- n. Scroll down to **Security Permissions** section and click **Add**.
- o. In the Add Security Permission dialog box, enter the values as shown in Figure 11-5.

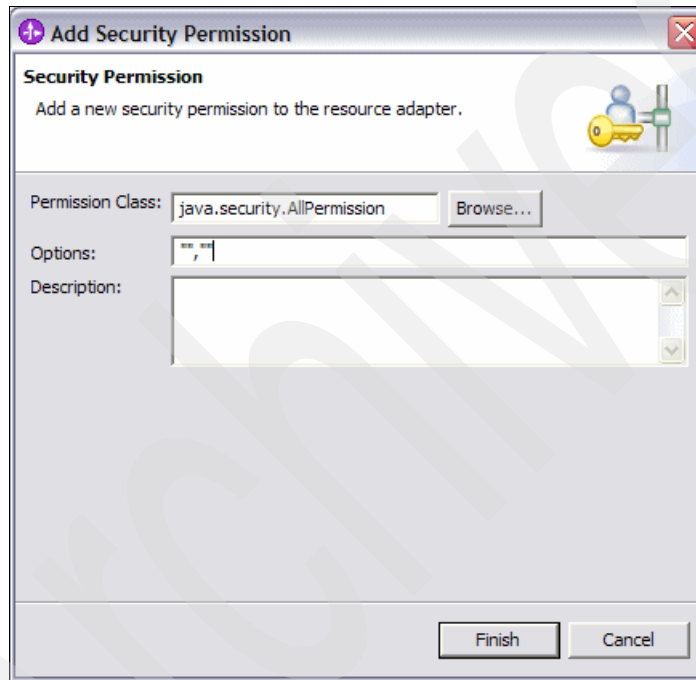


Figure 11-5 Add security permission for the adapter

Note: For simplicity, all permissions are granted to our adapter. In your implementation, you should set the appropriate security permissions for your adapter.

- p. Click **Finish**.
- q. Click **File** → **Save** to save the deployment descriptor.

11.4 Add dependent jar files

Our custom resource adapter has the following dependents jar file:

- ▶ RedMaintenance's common.jar
- ▶ Adapter foundation's CWYBS_AdapterFoundation.jar
- ▶ Internationalization libraries icu4j_3_2.jar

These files must be packaged inside the adapter .rar file when we deploy our adapter to WebSphere Process Server. Follow these steps so that they can be included in the adapter .rar file when we export our RedMaintenance module application:

1. Switch to **J2EE** perspective.
2. Import **RedMaintenance's common.jar** file into **Connector Project** → **RedMaintenance** → **connectorModule** directory.
3. Import **CWYBS_AdapterFoundation.jar** from C:\Program Files\IBM\ResourceAdapters\AdapterToolkit\adapter\base\lib into **Connector Project** → **RedMaintenance** → **connectorModule** directory.
4. Import **icu4j_3_2.jar** file from <WebSphere Integration Developer install location>\runtime\bi_v6\lib\icu4j_3_2.jar into **Connector Project** → **RedMaintenance** → **connectorModule** directory.
5. Right-click **RedMaintenance** connector project under Connector Project directory.
6. From the context menu, select **Properties**.
7. In the Properties for RedMaintenance dialog box, select **Java Build Path** → **Libraries**.
8. Select **CWYBS_AdapterFoundation.jar** external jar link.
9. Click **Remove**.
10. Click **Add Jars...**
11. In the Jar Selection dialog, select **common.jar**, **CWYBS_AdapterFoundation.jar** and **icu4j_3_2.jar** under **RedMaintenance** → **connectorModule** directory.
12. Click **OK**.
13. Click **OK**.

Note: The jar files added to our project are packaged inside our adapter.rar file along with other adapter files. We can now remove **CWYBS_AdapterFoundation.jar** file from *<WebSphere Process Server install location>\runtimes\bi_v6\lib* directory. This file was placed there when we created our HelloWorld adapter in Chapter 8, “Setting up the development environment” on page 159.

11.5 Implement connection to EIS

In Chapter 2, “Adapter basics” on page 33, we explain connection management contract specified under JCA specification. We also explain how an application component can get a connection to the EIS. Some of the goals of the managed connection contract are:

- ▶ Provide an efficient, scalable, and extensible connection pooling mechanism. This mechanism delegates the connection pooling management to the application server and frees the adapter developer from having to implement connection pooling.
- ▶ Provide a consistent programming model for applications to acquire connections to the EIS.

Figure 11-6 on page 263 shows an application component through SCA import connects to RedMaintenance server. It shows the main classes we need to implement in order to satisfy the resource adapter side of the JCA connection management contract. WebSphere Adapter Toolkit automatically generates these class stubs. These class stubs extends classes in WebSphere Adapter Foundation Classes.

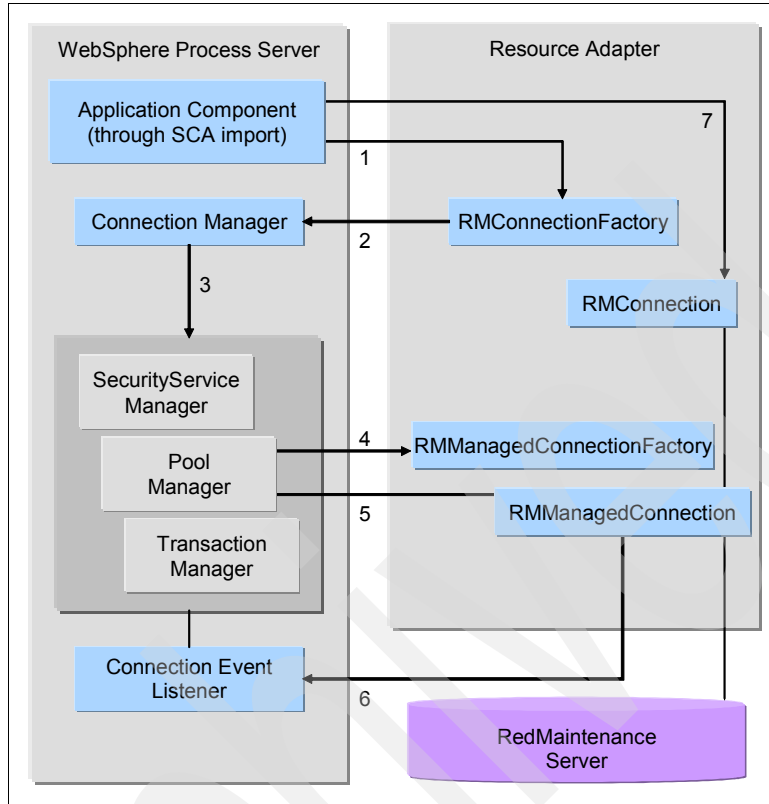


Figure 11-6 Main connection classes of the resource adapter

11.5.1 Implement RMConnection class

A `RMConnection` instance represents a application-level handle to the physical RedMaintenance connection. Implement the **`createInteraction`** method to return a new `RMInteraction` object. See Example 11-1.

Example 11-1 Implementation of `RMConnection` class

```
package com.ibm.itso.sab511.outbound;

import com.ibm.j2ca.base.WBICConnection;

public class RMConnection extends WBICConnection {
    /**
     * @param mgrdConn
     * @throws javax.resource.ResourceException
     */
    public RMConnection(com.ibm.j2ca.base.WBIManagedConnection mgrdConn)
```

```

        throws javax.resource.ResourceException {
            super(mgrdConn);
        }

    /**
     * @return a RedMaintenance specific Interaction implementation that can
     *         handle processing request on the RedMaintenance
     * @throws javax.resource.ResourceException
     */
    public javax.resource.cci.Interaction createInteraction()
        throws javax.resource.ResourceException {
        //verify this connection handle is still valid
        super.checkValidity();
        return new RMInteraction(this);
    }
}

```

11.5.2 Implement RMConnectionFactory class

RMConnectionFactory enables a application client to request handles to the underlying RedMaintenance connection. This class extends WBIConnectionFactory from the foundation classes. We can rely on the generic business logic provided in the parent class. There is no RedMaintenance-specific coding required in this class.

11.5.3 Implement RMManagedConnectionFactory class

A factory class to create physical managed connections to the underlying RedMaintenance application. Implementation the following methods:

- ▶ createConectionFactory
- ▶ createManagedConnection

See Example 11-2 for full implementation of this class. The fields rmiName, serverURL along with their getter and setter methods were created when we were adding them to the deployment descriptor.

Example 11-2 RMManagedConnectionFactory implementation

```

package com.ibm.itso.sab511.outbound;

import com.ibm.j2ca.base.WBIManagedConnectionFactory;

public class RMManagedConnectionFactory extends WBIManagedConnectionFactory {
    private String serverURL;

    private String rmiName;

```

```

private final String CLASS_NAME = RManagedConnectionFactory.class
    .getName();

/**
 *
 */
public RManagedConnectionFactory() {
    super();
}

/**
 * @return a new ManagedConnection instance to the EIS
 * @param subject - the authentication subject
 * @param conReqInfo - the connection request info object
 * @throws javax.resource.ResourceException
 */
public javax.resource.spi.ManagedConnection createManagedConnection(
    javax.security.auth.Subject subject,
    javax.resource.spi.ConnectionRequestInfo conReqInfo)
    throws javax.resource.ResourceException {
    final String METHOD_NAME = "createManagedConnection()";
    this.getLogUtils().traceMethodEntrance(CLASS_NAME, METHOD_NAME);
    RManagedConnection managedConnection = new RManagedConnection(this,
        subject, (RMConnectionRequestInfo) conReqInfo);
    this.getLogUtils().traceMethodExit(CLASS_NAME, METHOD_NAME);
    return managedConnection;
}

/**
 * @param connMgr - The connection manager in the app server
 * @return a connection factory instance
 */
public java.lang.Object createConnectionFactory(
    javax.resource.spi.ConnectionManager connMgr)
    throws javax.resource.ResourceException {
    return new RMConnectionFactory(connMgr, this);
}

public void setServerURL(String newValue) {
    String oldValue = this.serverURL;
    this.serverURL = newValue;
    super.getPropertyChangeSupport().firePropertyChange("serverURL",
        oldValue, newValue);
}

public String getServerURL() {
    return this.serverURL;
}

```

```

    public void setRmiName(String newValue) {
        String oldValue = this.rmiName;
        this.rmiName = newValue;
        super.getPropertyChangeSupport().firePropertyChange("rmiName",
            oldValue, newValue);
    }

    public String getRmiName() {
        return this.rmiName;
    }
}

```

11.5.4 Implement RManagedConnection class

This class represents a managed physical connection to the RedMaintenance application and the physical connection to RedMaintenance is implemented in this class. You must also implement the destroy method to facilitate connection closure. See Example 11-3.

Example 11-3 RManagedConnection implementation

```

package com.ibm.itso.sab511.outbound;

import java.rmi.Naming;
import java.util.logging.Level;

import javax.resource.spi.security.PasswordCredential;

import com.ibm.itso.rm.common.RMServerRMIInterface;
import com.ibm.j2ca.base.WBIManagedConnection;
import com.ibm.j2ca.base.WBIManagedConnectionMetaData;
import com.ibm.j2ca.extension.logging.LogUtilConstants;
import com.ibm.j2ca.extension.logging.LogUtils;

public class RManagedConnection extends WBIManagedConnection {
    private RMServerRMIInterface eisConnection;

    private LogUtils logUtils;

    private final String CLASS_NAME = RManagedConnection.class.getName();

    private String serverURL;

    private String rmiName;

    /**
     * @param managedConnectionFactory

```

```

    * @param subject
    * @param connReqInfo
    * @throws javax.resource.ResourceException
    */
    public RMManagedConnection(
        com.ibm.j2ca.base.WBIManagedConnectionFactory
managedConnectionFactory,
        javax.security.auth.Subject subject,
        com.ibm.j2ca.base.WBIConnectionRequestInfo connReqInfo)
        throws javax.resource.ResourceException {
        super(managedConnectionFactory, subject,
            (RMConnectionRequestInfo) connReqInfo);
        this.logUtils = getLogUtils();
        this.serverURL = ((RMManagedConnectionFactory) managedConnectionFactory)
            .getServerURL();
        this.rmiName = ((RMManagedConnectionFactory) managedConnectionFactory)
            .getRmiName();
        logUtils.trace(Level.FINER, CLASS_NAME, "Constructor",
            "creating managed connection");
    }

    /**
     * @param credential
     * @return a physical connection to RedMaintenance
     */
    private boolean connectToEIS(PasswordCredential credential) {
        boolean connected = false;
        String rmiURL = "/" + this.serverURL + "/" + this.rmiName;

        try {
            eisConnection = (RMServerRMIInterface) Naming.lookup(rmiURL);
            connected = true;
        } catch (Exception e) {
            logUtils.log(Level.SEVERE, LogUtilConstants.ADAPTER_RBUNDLE,
                CLASS_NAME, "connectToEIs", "1001", new Object[] {});
        }
        return connected;
    }

    /**
     * @return information about the EIS connection
     */
    public javax.resource.spi.ManagedConnectionMetaData getMetaData()
        throws javax.resource.ResourceException {
        String eisProductName = "RedMaintenance";
        String eisProductVersion = "1.0";
        int maxConnections = 5;
        String userName = super.getPasswordCredential().getUserName();
        return new WBIManagedConnectionMetaData(eisProductName,

```

```

        eisProductVersion, maxConnections, userName);
    }

    /**
     * @return a RMConnection instance associated with the underlying EIS
     *         connection
     */
    public java.lang.Object getWBICConnection(
        javax.resource.spi.security.PasswordCredential credential,
        boolean arg1) throws javax.resource.ResourceException {
        //Create a new EIS connection only if one has not been already created
        if (eisConnection == null) {
            this.connectToEIS(credential);
        }
        return new RMConnection(this);
    }

    /**
     * Close the physical connection to the EIS and release any resources
     */
    public void destroy() throws javax.resource.ResourceException {
        //set rmi connection to RedMaintenance to null
        eisConnection = null;
    }

    /**
     * @return Returns the eisConnection.
     */
    public RMServerRMIInterface getEisConnection() {
        return eisConnection;
    }
}

```

11.6 Implement outbound operations

Outbound request processing enables application service components running on WebSphere Process Server to execute operations on RedMaintenance through the resource adapter. The result of the operations are returned to the application service component. The data exchanged between the application component and the resource adapter are packaged in application-specific business objects.

As described in Chapter 2, “Adapter basics” on page 33, one of the reasons for having a JCA compliant resource adapter is to standardize the interaction between application components and EISs. By following standards, we can reduce integration complexity between M application and N EIS from M x N to M

+ N. Under the JCA specification, outbound processing uses the following standard programming model to interact between application component and resource adapter:

1. CCI client, for example an EJB or other business process, uses JNDI service to look up a connection factory for the adapter.
2. Request a connection from the connection factory.
3. Use the Connection object to create an Interaction object.
4. Create an **InteractionSpec** object. Specify the function to be executed in the InteractionSpec object.
5. Create a **Record** object. This Record object is used to pass data into the EIS.
6. Execute the function through the interaction object.
7. The function execution returns data to the client through a Record object.
8. Close the **Interaction**.
9. Close the **Connection**.

In 11.3, “Generate outbound stub classes” on page 255 section, we select to generate an IBM WebSphere Resource Adapter type adapter project. For this type of project, WebSphere Adapter Toolkit automatically generates stub classes that extend WebSphere Foundation Classes. WebSphere Foundation Classes include generic implementation of JCA contracts and methods that make it easier to develop a custom resource adapter. Some of the interaction steps described here are handled by the foundation classes. WebSphere Foundation Classes also provides the Command Pattern to help business object processing. The Command Pattern has generic business logic to help process business objects. See 4.11, “Command Pattern” on page 95 for details. The use of Command Pattern is not required, but we strongly recommend using it. To use the Command Pattern for our custom adapter development, we must implement the following stub classes generated by WebSphere Adapter Toolkit:

1. RMCommandFactoryImpl

This factory class is used to create a specific Command object for a given input function name. For example, if the function name is Create then the command class created is RMCreateCommand.

2. RMBaseCommand

This is the parent of the following command classes. It is good programming practice to put common command business logic here so they can be utilized by the children command classes.

3. RMCreateCommand

Implement business logic in this class to create a new RedMaintenance record.

4. RMUpdateCommand

Implement business logic in this class to update a RedMaintenance record.

5. RMRetrieveCommand

For a given key, this class must implement business logic to retrieve the corresponding RedMaintenance record and all of its off spring RedMaintenance records recursively. For example, if the adapter is given an Apartment key, it must retrieve:

- a. The Apartment record associated with this key
- b. Any Maintenance records associated with the above Apartment
- c. Any PartOrder records associated with the above Maintenance record

6. RMDeleteCommand

For a given key, this class must implement business logic to delete the corresponding RedMaintenance record and all of its off spring RedMaintenance record recursively. Many EIS's delete API handles the recursive deletion of children, but RedMaintenance delete API does not. We must handle this logic in our RMDeleteCommand class.

11.6.1 Implement RMInteraction class

RMInteraction class provides an execute method. The business object sent from the application component is passed to this method as a Record object. A RMInteractionSpec object is also passed into the execute method. The execute method extracts the function or operation name from the RMInteractionSpec object. The command manager of the Command Pattern takes the function name and the input Record object to create a Command object. The Command Pattern's Interpreter object then process the Command object.

This processing involves traversing the input data object tree and invoking the corresponding commands on each node of the input data object hierarchy. While it is processing the input business object tree, it builds the response data object hierarchy. At the end of the processing a populated data object hierarchy is returned. The last step in the execute method is to return the data object as a Record object to comply with JCA specification.

See implementation of RMInteraction class in Example 11-4 on page 271.

Note: If you copy the implementation for this class into your connector project, you would see some errors in the problems view. These errors disappear when you implement ObjectNaming class and RMCommandFactoryImpl classes.

Example 11-4 Implementation of *RMInteraction* class

```
package com.ibm.itso.sab511.outbound;

import javax.resource.ResourceException;

import com.ibm.itso.rm.common.RMServerRMIInterface;
import com.ibm.itso.sab511.RMResourceAdapter;
import com.ibm.itso.sab511.outbound.commandpattern.RMCommandFactoryImpl;
import com.ibm.itso.sab511.utils.ObjectNaming;
import com.ibm.j2ca.base.WBIInteraction;
import com.ibm.j2ca.base.WBIInteractionSpec;
import com.ibm.j2ca.base.WBIRecord;
import com.ibm.j2ca.extension.commandpattern.Command;
import com.ibm.j2ca.extension.commandpattern.CommandManager;
import com.ibm.j2ca.extension.commandpattern.Interpreter;
import com.ibm.j2ca.extension.logging.LogUtils;
import commonj.sdo.DataObject;

public class RMInteraction extends WBIInteraction {

    private RMServerRMIInterface eisConnection;

    private CommandManager commandManager;

    private Interpreter interpreter;

    private LogUtils logger;

    private RMCommandFactoryImpl factory;

    /**
     * @param connection
     * @throws ResourceException
     */
    public RMInteraction(com.ibm.j2ca.base.WBIConnection connection)
        throws ResourceException {
        super(connection);
        RMManagedConnection managedConnection = (RMManagedConnection) connection
            .getManagedConnection();
        eisConnection = managedConnection.getEisConnection();

        logger = this.getLogUtils();
        interpreter = new Interpreter(this.getLogUtils());
        RMResourceAdapter resourceAdapter = (RMResourceAdapter)
managedConnection
            .getManagedConnectionFactory().getResourceAdapter();
        ObjectNaming objectNaming = new ObjectNaming(resourceAdapter);
        factory = new RMCommandFactoryImpl(objectNaming);
    }
}
```

```

        commandManager = new CommandManager(factory, eisConnection, this
            .getLogUtils());
    }

    /**
     * Performs an operation on RedMaintenance according to the function name
     * specified in RMInteractionSpec. It takes a Record from the client and
     * returns a Record to the client.
     *
     * @param ispec - a RMInteractionSpec objec
     * @param inRecord - the data object from the client
     * @throws javax.resource.ResourceException
     */
    public javax.resource.cci.Record execute(
        javax.resource.cci.InteractionSpec ispec,
        javax.resource.cci.Record inRecord)
        throws javax.resource.ResourceException {
        WBIInteractionSpec interactionSpec = (WBIInteractionSpec) ispec;
        String functionName = interactionSpec.getFunctionName();
        Command topLevelCommand = commandManager.produceCommands(
            (WBIRecord) inRecord, functionName);
        DataObject returnDataObject;
        returnDataObject = interpreter.execute(topLevelCommand);

        WBIRecord outRecord = new WBIRecord();
        if (functionName == WBIInteractionSpec.RETRIEVE_ALL_OP) {
            outRecord.setDataObject(returnDataObject);
        } else {
            outRecord.setDataObject(returnDataObject.getContainer());
        }
        logger.traceMethodExit(RMInteraction.class.getName(), "execute()");
        return outRecord;
    }
}

```

11.6.2 Implement RMInteractionSpec class

In this class, we simply create a pair of getter and setter methods for function name. These methods call the parent getter and setter method for function name. This might seem redundant, however adding this method pair helps resolve errors and warnings in the SCA import file when we create it later. The calling application component does not need to physically set the function name. The function name is specified in SCA import file.

See implementation of this class in Example 11-5 on page 273.

Example 11-5 RMInteractionSpec implementation

```
package com.ibm.itso.sab511.outbound;

import com.ibm.j2ca.base.WBIInteractionSpec;

public class RMInteractionSpec extends WBIInteractionSpec {

    public RMInteractionSpec() {
        super();
    }

    public String getFunctionName() {
        return super.getFunctionName();
    }

    public void setFunctionName(String arg0) {
        super.setFunctionName(arg0);
    }
}
```

11.6.3 Create ObjectNaming utility class

ObjectNaming is a utility class for data objects. It has methods to gather the data object's meta information. WebSphere Adapter Toolkit does not generate a stub code for this class. When creating this class in our connector project, we specify the package as: `com.ibm.itso.sab511.utils`. See implementation in Example 11-6.

Example 11-6 ObjectNaming implementation

```
package com.ibm.itso.sab511.utils;

import java.util.HashMap;
import java.util.Iterator;
import java.util.logging.Level;

import javax.resource.ResourceException;

import com.ibm.itso.sab511.RMResourceAdapter;
import com.ibm.j2ca.base.AdapterBOUtil;
import com.ibm.j2ca.extension.logging.LogUtils;
import commonj.sdo.DataObject;
import commonj.sdo.Property;

/**
 * ObjectNaming
 */
```

```

    * This class is intended to read metadata from business objects, and to be
able
    * to set and unset keys inside business objects.
    */
public class ObjectNaming {
    private static final String FIELDNAME = "FieldName";
    private static final String OBJECTNAME = "ObjectName";
    private LogUtils logger;
    private RMResourceAdapter resourceAdapter;

    public ObjectNaming(RMResourceAdapter resourceAdapter) {
        logger = resourceAdapter.getLogUtils();
        this.resourceAdapter = resourceAdapter;
    }

    /**
     * getEntityName - get the Native/RedMaintenance name of a given DataObject
     *
     * @param dataObject -
     *             the dataObject we want to get the name of
     * @return the object's name
     * @throws ResourceException
     */
    public String getEntityName(DataObject dataObject) throws ResourceException
    {
        DataObject entityInfo = null;
        logger.trace(Level.FINEST, ObjectNaming.class.getName(), "getEntityName()",
            "Attempting to get entity name of do " + dataObject.getType().getName());
        entityInfo = AdapterBOUtil.getMetadataForObject(dataObject);
        String name = entityInfo.getString(OBJECTNAME);
        logger.trace(Level.FINEST, ObjectNaming.class.getName(),
            "getEntityName()", "Name is " + name);
        return name;
    }

    /**
     * getRealColumnNames
     *
     * @param dataObject
     *             the dataObject we want the column names of
     * @return a hashmap that maps the dataObject attribute's name to the
     *             native/RedMaintenance name of that column
     * @throws ResourceException
     */
    public HashMap getRealColumnNames(DataObject dataObject) throws
ResourceException {
        HashMap attributeASI = new HashMap();
        Iterator attributeIterator =
dataObject.getType().getProperties().iterator();

```

```

        while (attributeIterator.hasNext()) {
            Property thisProperty = (Property) attributeIterator.next();
            String propertyName = thisProperty.getName();
            logger.trace(Level.FINEST, ObjectNaming.class.getName(),
                "getRealColumnNames()", "Attempting to get column name of attribute: " +
                propertyName);
            DataObject attributeDO =
                AdapterBOUtil.getMetadataForProperty(thisProperty);
            if (attributeDO != null) {
                String columnName = attributeDO.getString(FIELDNAME);
                //attributeASI.put(propertyName, columnName.toUpperCase());
                attributeASI.put(propertyName, columnName);
            }
        }
        return attributeASI;
    }

    /**
     * getKey - get the key of this data object as a string.
     *
     * @param dataObject
     *         -the dataObject for which we want the key
     * @return the key
     * @throws ResourceException
     */
    public String getKey(DataObject dataObject) throws ResourceException {
        Property[] keyProperties = AdapterBOUtil.getKeyProperties(dataObject);
        return dataObject.getString(keyProperties[0]);
    }

    /**
     * setKey - set the key of a given data object
     *
     * @param dataObject
     * @param key
     * @throws ResourceException
     */
    public void setKey(DataObject dataObject, String key) throws ResourceException
    {
        Property[] keyProperties = AdapterBOUtil.getKeyProperties(dataObject);
        dataObject.setString(keyProperties[0], key);
    }

    /**
     * unSetKey - unset the key
     *
     * @param dataObject
     * @throws ResourceException
     */

```

```

        public void unSetKey(DataObject dataObject) throws ResourceException {
            Property[] keyProperties = AdapterBOUtil.getKeyProperties(dataObject);
            dataObject.unset(keyProperties[0]);
        }

        /**
         * @return Returns the logger.
         */
        public LogUtils getLogger() {
            return logger;
        }

        /**
         * @param logger
         *         The logger to set.
         */
        public void setLogger(LogUtils logger) {
            this.logger = logger;
        }

        /**
         * @return Returns the resourceAdapter.
         */
        public RMResourceAdapter getResourceAdapter() {
            return resourceAdapter;
        }

        /**
         * @param resourceAdapter
         *         The resourceAdapter to set.
         */
        public void setResourceAdapter(RMResourceAdapter resourceAdapter) {
            this.resourceAdapter = resourceAdapter;
        }
    }
}

```

11.6.4 Create ObjectConverter utility class

ObjectConverter is a utility class to convert incoming data object to RedMaintenance object and vice versa. See Example 11-7. This class is not generated by WebSphere Adapter Toolkit. When creating class in our connector project, we specify the package as: com.ibm.itso.sab511.utils.

Example 11-7 ObjectConvert implementation

```

package com.ibm.itso.sab511.utils;

import java.util.HashMap;

```

```

import java.util.Iterator;
import java.util.List;
import java.util.logging.Level;

import javax.resource.ResourceException;

import com.ibm.itso.rm.common.RMDataImplementation;
import com.ibm.itso.sab511.RMResourceAdapter;
import com.ibm.j2ca.extension.logging.LogUtils;
import commonj.sdo.DataObject;
import commonj.sdo.Property;
import commonj.sdo.Type;

/**
 * ObjectConverter. This class can convert an RedMaintenance object to a
 * DataObject and vice-versa.
 */
public class ObjectConverter {
    ObjectNaming objectNaming;

    LogUtils logger;

    RMResourceAdapter ra;

    /**
     * Constructor for ObjectSerializer
     *
     * @param objectNaming
     */
    public ObjectConverter(ObjectNaming objectNaming) {
        this.objectNaming = objectNaming;
        this.logger = objectNaming.getLogger();
        this.ra = objectNaming.getResourceAdapter();
    }

    /**
     * toRedMaintenanceObject - convert the dataObject to a RedMaintenance
     * object
     *
     * @param dataObject
     * @return the RedMaintenance object
     * @throws ResourceException
     */
    public RMDataImplementation toRedMaintenanceObject(DataObject dataObject)
        throws ResourceException {
        logger.trace(Level.FINER, ObjectConverter.class.getName(),
            "toRMDataImplementation", "Convert data object: "
                + dataObject.getType().getName());
        RMDataImplementation redMaintenanceObject = new RMDataImplementation();
    }

```

```

        List propertyList = dataObject.getType().getProperties();
        HashMap realColumnNames = objectNaming.getRealColumnNames(dataObject);
        Iterator propertyIterator = propertyList.iterator();
        while (propertyIterator.hasNext()) {
            Property currentProperty = (Property) propertyIterator.next();
            if (currentProperty.isContainment() || currentProperty.isMany())
                continue;
            String propertyName = currentProperty.getName();
            Object currentValue = dataObject.getString(propertyName);
            String EISPropertyName = ((String) realColumnNames
                .get(propertyName)).trim();
            if (currentValue != null) {
                redMaintenanceObject.setAttr(EISPropertyName, currentValue
                    .toString());
            }
        }
        return redMaintenanceObject;
    }

/**
 * toDataObject - convert the RedMaintenance object to a data object.
 *
 * @param redMaintenanceObject -
 *         the input RedMaintenance object
 * @param dataObject -
 *         an empty data object to populate.
 * @return the data object populated with the correct values
 * @throws ResourceException
 */
public DataObject toDataObject(RMDataImplementation redMaintenanceObject,
    DataObject dataObject) throws ResourceException {
    HashMap propertiesToRealColumns = objectNaming
        .getRealColumnNames(dataObject);
    Type type = dataObject.getType();
    List propertyList = type.getProperties();
    Iterator propertyIterator = propertyList.iterator();
    while (propertyIterator.hasNext()) {
        Property property = (Property) propertyIterator.next();
        String propertyName = property.getName();
        String EISPropertyName = ((String) propertiesToRealColumns
            .get(propertyName)).trim();
        String value = redMaintenanceObject.getAttr(EISPropertyName);
        if ((value != null) && (!(value.equalsIgnoreCase("null")))) {
            dataObject.setString(propertyName, value);
        }
    }
    return dataObject;
}

```



```
}
```

Important: Business object definition contains application-specific information (ASI) to help the adapter process the incoming business object. Business object definitions are stored as XML schemas in .xsd files. Adapter foundation classes provides utility classes to parse application-specific informations. Given ASI is stored in a XML file, the data between tags might contain line breaks and blank spaces. Make sure to call the trim method when getting ASI information from business objects. For example:

```
String EISPropertyName = ((String)
propertiesToRealColumns.get(propertyName)).trim();
```

11.6.5 Implement RMCommandFactoryImpl class

This class is required by the Command Pattern to return a Command object that correspond to the input function name. For example, if the function name is Create, the createCommand method of this class returns a RMCreateCommand object.

Note: You might see an error in your problems view if you paste this code into your connector project. This error disappears when you implement RMBaseCommand class.

Important: If you want the Command Pattern to recursively process the incoming business object, you must return true from isOOType method. It returns false by default. When isOOType returns false, only the top-level node is processed by the Command Pattern.

See Example 11-8.

Example 11-8 RMCommandFactoryImpl implementation

```
package com.ibm.itso.sab511.outbound.commandpattern;

import com.ibm.itso.sab511.utils.ObjectConverter;
import com.ibm.itso.sab511.utils.ObjectNaming;
import com.ibm.j2ca.extension.commandpattern.Command;
import com.ibm.j2ca.extension.commandpattern.CommandFactory;
import com.ibm.j2ca.extension.commandpattern.NodeLevelOperations;

public class RMCommandFactoryImpl implements CommandFactory {

    private ObjectNaming objectNaming;
```

```

private ObjectConverter objectConverter;

public RMCommandFactoryImpl(ObjectNaming objectNaming) {
    this.objectNaming = objectNaming;
    objectConverter = new ObjectConverter(objectNaming);
}

/**
 * Return the corresponding command for a given function name
 *
 */
public com.ibm.j2ca.extension.commandpattern.Command createCommand(
    java.lang.String functionName, commonj.sdo.DataObject dataObject)
    throws javax.resource.ResourceException {
    RMBaseCommand command = null;
    if (functionName.equals(NodeLevelOperations.CREATE_NODE)) {
        command = new RMCreateCommand(dataObject);
    } else if (functionName.equals(NodeLevelOperations.DELETE_NODE)) {
        command = new RMDeleteCommand(dataObject);
    } else if (functionName.equals(NodeLevelOperations.UPDATE_NODE)) {
        command = new RMUpdateCommand(dataObject);
    } else if (functionName.equals(NodeLevelOperations.RETRIEVE_STRUCTURE))
    {
        command = new RMRetrieveCommand(dataObject);
    } else if (functionName.equals(NodeLevelOperations.RETRIEVE_ALL)) {
        command = new RMRetrieveAllCommand(dataObject);
    } else {
        command = new RMBaseCommand(dataObject);
    }

    command.setObjectConverter(objectConverter);
    command.setObjectNaming(objectNaming);

    if (functionName == NodeLevelOperations.DELETE_NODE) {
        command.setExecutionOrder(Command.BEFORE_PARENT);
    } else {
        command.setExecutionOrder(Command.AFTER_PARENT);
    }
    return command;
}

/**
 * Specify whether you want the command manager to set command to only the
 * top level object or set commands for children objects as well.
 *
 * @return Return false if command manager should only set the top level
 *         object. Return true if you want child objects to receive
 *         commands
 */

```

```

        *           as well.
        */
        public boolean isOOType() {
            return true;
        }
    }
}

```

11.6.6 Implement RMBaseCommand class

This is the parent class for all RedMaintenance command classes. It extends the Command class of the CommandPattern. We put common command business logic in this class. See Example 11-9.

Example 11-9 RMBaseCommand implementation

```

package com.ibm.itso.sab511.outbound.commandpattern;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.itso.sab511.utils.ObjectConverter;
import com.ibm.itso.sab511.utils.ObjectNaming;
import com.ibm.j2ca.extension.commandpattern.Command;
import commonj.sdo.DataObject;
import commonj.sdo.Property;

public class RMBaseCommand extends Command {
    private ObjectNaming objectNaming;
    private ObjectConverter objectConverter;

    public RMBaseCommand() {
        super();
    }

    public RMBaseCommand(commonj.sdo.DataObject arg0) {
        super(arg0);
    }

    public commonj.sdo.DataObject execute(commonj.sdo.DataObject inputObject)
        throws javax.resource.ResourceException {
        return inputObject;
    }

    /**
     * @return Returns the objectConverter.
     */
    public ObjectConverter getObjectConverter() {

```

```

        return objectConverter;
    }
    /**
     * @param objectConverter The objectConverter to set.
     */
    public void setObjectConverter(ObjectConverter objectConverter) {
        this.objectConverter = objectConverter;
    }
    /**
     * @return Returns the objectNaming.
     */
    public ObjectNaming getObjectNaming() {
        return objectNaming;
    }
    /**
     * @param objectNaming The objectNaming to set.
     */
    public void setObjectNaming(ObjectNaming objectNaming) {
        this.objectNaming = objectNaming;
    }
    /**
     * @param dataObject
     * @return a list of children names
     */
    protected List findChildAttributes(DataObject dataObject) {
        List childAttributes = new ArrayList();
        if (dataObject == null)
            return null;
        List propertyList = dataObject.getType().getProperties();
        Iterator propertyIterator = propertyList.iterator();
        while (propertyIterator.hasNext()) {
            Property currentProperty = (Property) propertyIterator.next();
            if (currentProperty.isContainment() || currentProperty.isMany()) {
                String propertyName = currentProperty.getName();
                childAttributes.add(propertyName);
            }
        }
        return childAttributes;
    }
}

```

11.6.7 Implement RMCreatCommand class

This class implements logic for invoking RedMaintenance's API to create a new RedMaintenance record. It does this as follows:

1. Link current object to parent object. This sets a foreign key in the child object to point to the parent object. This is necessary to preserve parent and child relationship.
2. Gather metadata information of the incoming data object.
3. Create a new **RedMaintenance** object from the metadata.
4. Invoke RedMaintenance's **createObject** method.
5. Retrieve the key for the newly created record.
6. Set incoming data object with new key.
7. Return incoming data object.

This class does not need to be concerned if the incoming business object is a hierarchical business object. It only must implement the logic to handle a single node. The Command Pattern is responsible to traverse the incoming business object hierarchy to create new RedMaintenance records for all nodes in the incoming business object hierarchy tree. See Example 11-10.

Example 11-10 RMCreatCommand implementation

```
package com.ibm.itso.sab511.outbound.commandpattern;

import java.rmi.RemoteException;
import java.util.logging.Level;

import javax.resource.ResourceException;
import javax.resource.spi.CommException;

import com.ibm.itso.rm.common.RMDataImplementation;
import com.ibm.itso.rm.common.RMDataInterface;
import com.ibm.itso.rm.common.RMServerRMIInterface;
import com.ibm.itso.sab511.utils.ObjectNaming;
import com.ibm.j2ca.extension.logging.LogUtilConstants;
import commonj.sdo.DataObject;

public class RMCreatCommand extends RMBaseCommand {

    RMServerRMIInterface connection;

    public RMCreatCommand() {
        super();
    }
}
```

```

public RMCreateCommand(commonj.sdo.DataObject inputObject) {
    super(inputObject);
}

/**
 * Call RedMaintenance API to create a new RedMaintenance record
 *
 * @return Return a data object that is a "snap shot" image of the newly
 *         create record.
 */
public DataObject execute(DataObject inputObject) throws ResourceException
{
    getLogUtils().traceMethodEntrance(RMCreateCommand.class.getName(),
        "execute()");
    ObjectNaming objectNaming = super.getObjectNaming();
    String entity = objectNaming.getEntityName(inputObject);
    connection = (RMServerRMIInterface) this.getConnection();
    objectNaming.unSetKey(inputObject);
    RMDDataImplementation requestRMOBJECT = super.getObjectConverter()
        .toRedMaintenanceObject(inputObject);
    DataObject parent = inputObject.getContainer();
    if (parent.getContainer() == null) {
        parent = null;
    }
    linkChildToParent(parent, objectNaming, requestRMOBJECT, inputObject);
    String key = null;

    requestRMOBJECT.setComponentName(entity);
    requestRMOBJECT.setExecutionType("Create");

    try {
        RMDDataInterface responseRMOBJECT = connection
            .createObject(requestRMOBJECT);
        key = responseRMOBJECT.getAttr("Id");
    } catch (RemoteException e) {
        getLogUtils().log(Level.SEVERE, LogUtilConstants.ADAPTER_RBUNDLE,
            RMCreateCommand.class.getName(), "execute()", "1002",
            new Object[] {});
        throw new CommException(e);
    }

    objectNaming.setKey(inputObject, key);
    getLogUtils().traceMethodExit(RMCreateCommand.class.getName(),
        "execute()");
    return inputObject;
}

/**
 * Create a foreign key in the child object to point to the parent object

```

```

    * @param parentObject
    * @param objectNaming
    * @param redMaintenanceObject
    */
    private void linkChildToParent(DataObject parentObject,
        ObjectNaming objectNaming,
        RMDDataImplementation redMaintenanceObject, DataObject inputObject)
        throws ResourceException {
        if (parentObject != null) {
            String parentKey = objectNaming.getKey(parentObject);
            String parentName = objectNaming.getEntityName(parentObject);
            String parentAttribute = parentName + "Id";
            redMaintenanceObject.setAttr(parentAttribute, parentKey);
            inputObject.unset(parentAttribute);
            inputObject.setString(parentAttribute, parentKey);
        }
    }
}

```

11.6.8 Implement RMUpdateCommand class

This class implements the logic for invoking RedMaintenance's API to update a RedMaintenance record. It does this by doing the following:

1. Gather metadata information of the incoming data object.
2. Create a new RedMaintenance object from the metadata.
3. Invoke RedMaintenance's updateObject method.
4. Return incoming data object.

This class does not need to be concerned if the incoming business object is a hierarchical business object. It only must implement the logic to handle a single node. The Command Pattern is responsible to traverse the incoming business object hierarchy to update RedMaintenance records for all nodes in the incoming business object hierarchy tree. See Example 11-11.

Example 11-11 RMUpdateCommand implementation

```

package com.ibm.itso.sab511.outbound.commandpattern;

import java.rmi.RemoteException;
import java.util.logging.Level;

import javax.resource.ResourceException;
import javax.resource.spi.CommException;

import com.ibm.itso.rm.common.RMDDataImplementation;
import com.ibm.itso.rm.common.RMServerRMIInterface;

```

```

import com.ibm.j2ca.extension.logging.LogUtilConstants;
import commonj.sdo.DataObject;

public class RMUpdateCommand extends RMBaseCommand {

    public RMUpdateCommand() {
        super();
    }

    public RMUpdateCommand(commonj.sdo.DataObject arg0) {
        super(arg0);
    }

    /**
of  * Updates one record in RedMaintenance. Command Pattern handles the update
    * input business object hierachy, this method only need to concern with
    * updating a single record in RedMaintenance.
    */
    public DataObject execute(DataObject inputRecord) throws ResourceException
    {
        getLogUtils().traceMethodEntrance(RMUpdateCommand.class.getName(),
            "execute()");
        String entity = super.getObjectNaming().getEntityName(inputRecord);
        String key = super.getObjectNaming().getKey(inputRecord);
        RMServerRMIInterface connection = (RMServerRMIInterface) this
            .getConnection();
        RMDDataImplementation redMaintenanceObject = super.getObjectConverter()
            .toRedMaintenanceObject(inputRecord);
        redMaintenanceObject.setComponentName(entity);
        redMaintenanceObject.setExecutionType("Update");
        try {
            connection.updateObject(redMaintenanceObject);
        } catch (RemoteException e) {
            getLogUtils().log(Level.SEVERE, LogUtilConstants.ADAPTER_RBUNDLE,
                RMCCreateCommand.class.getName(), "execute()", "1004",
                new Object[] { key });
            throw new CommException(e);
        }
        getLogUtils().traceMethodEntrance(RMUpdateCommand.class.getName(),
            "execute()");
        return inputRecord;
    }
}

```

11.6.9 Implement RMRetrieveCommand class

This class implements business logic to retrieve recursively the RedMaintenance record hierarchy for a given parent key in the data object. Because only a parent key is given, in order to know if the parent has any off spring this class must rely on the application-specific information provided in the business object definition for parent child relationship information. For example, the Apartment business object definition must define that it can contain an array of Maintenance business objects. See implementation in Example 11-12.

Example 11-12 RMRetrieveCommand implementation

```
package com.ibm.itso.sab511.outbound.commandpattern;

import java.rmi.RemoteException;
import java.util.Iterator;
import java.util.List;
import java.util.Vector;
import java.util.logging.Level;
import javax.resource.ResourceException;
import javax.resource.spi.CommException;
import com.ibm.itso.rm.common.RMDataImplementation;
import com.ibm.itso.rm.common.RMDataInterface;
import com.ibm.itso.rm.common.RMServerRMIInterface;
import com.ibm.j2ca.base.exceptions.RecordNotFoundException;
import com.ibm.j2ca.extension.logging.LogUtilConstants;
import commonj.sdo.DataObject;

public class RMRetrieveCommand extends RMBaseCommand {
    RMServerRMIInterface connection;

    public RMRetrieveCommand() {
        super();
    }

    public RMRetrieveCommand(commonj.sdo.DataObject arg0) {
        super(arg0);
    }

    /**
     * This method is called by the Command Pattern's Interpreter to retrieve
     * records in RedMaintenance.
     */
    public DataObject execute(DataObject inputObject) throws ResourceException
    {
        getLogUtils().traceMethodEntrance(RMRetrieveCommand.class.getName(),
            "execute()");
        String entity = super.getObjectNaming().getEntityName(inputObject);
        DataObject dataObject;
```

```

String key = super.getObjectNaming().getKey(inputObject);
try {
    connection = (RMServerRMIInterface) this.getConnection();
    dataObject = populateThisNode(inputObject, entity, key);
    if (dataObject != null) {
        populateChildren(dataObject, entity, key);
    }
} catch (RemoteException e) {
    getLogUtils().log(Level.SEVERE, LogUtilConstants.ADAPTER_RBUNDLE,
        RMRetrieveCommand.class.getName(), "execute()", "1005",
        new Object[] { key });
    throw new CommException(e);
}
getLogUtils().traceMethodExit(RMRetrieveCommand.class.getName(),
    "execute()");
return inputObject;
}

/**
 * Populate all children nodes for the business object being retrieved.
 *
 * @param dataObject
 * @param entity
 * @param key
 * @throws RemoteException
 * @throws ResourceException
 */
private void populateChildren(DataObject dataObject, String entity,
    String key) throws RemoteException, ResourceException {
    if (dataObject == null)
        return;
    getLogUtils().trace(Level.FINEST, RMRetrieveCommand.class.getName(),
        "populateChildren()",
        "attempting to populate children in " + entity + "." + key);
    List childAttributes = findChildAttributes(dataObject);
    Iterator childAttributeIterator = childAttributes.iterator();
    while (childAttributeIterator.hasNext()) {
        String currentChildName = (String) childAttributeIterator.next();
        RMDataImplementation requestObject = new RMDataImplementation();
        requestObject.setComponentName(currentChildName);
        requestObject.setAttr(entity + "Id", key);
        try {
            RMDataInterface responseObject = connection
                .retrieveObjectByValue(requestObject);
            Vector childObjects = responseObject.getRetrieveByValue();
            if (childObjects != null) {
                Vector childKeys = new Vector();
                for (int i = 0; i < childObjects.size(); i++) {

```

```

        RMDDataInterface childInstance = (RMDDataInterface)
childObjects
            .get(i);
        if (!childInstance.getAttr("Status").equals("D")) {
            childKeys.add(childInstance.getAttr("Id"));
        }
    }
    populateAttributeContainingChild(dataObject, childKeys,
        currentChildName);
}
} catch (NullPointerException e) {
    //DataObject has no children, recursion stops here.
    getLogUtils().trace(
        Level.FINE,
        RMRetrieveCommand.class.getName(),
        "populateChildren()",
        super.getObjectNaming().getEntityName(dataObject)
            + " has no children");
}
}
}

private void populateAttributeContainingChild(DataObject parentObject,
    Vector childKeys, String currentChildName) throws RemoteException,
    ResourceException {
    getLogUtils().trace(
        Level.FINEST,
        RMRetrieveCommand.class.getName(),
        "populateAttributeContainingChild()",
        "attempting to populate child container " + currentChildName
            + "." + childKeys.toString());
    Iterator childKeyIterator = childKeys.iterator();
    while (childKeyIterator.hasNext()) {
        String currentChildKey = (String) childKeyIterator.next();
        DataObject childObject = parentObject
            .createDataObject(currentChildName);
        DataObject currentObject = populateThisNode(childObject,
            currentChildName, currentChildKey);
        populateChildren(currentObject, currentChildName, currentChildKey);
    }
}

/**
 * @param inputObject
 * @param entity
 * @param key
 * @param connection
 * @throws ResourceException

```

```

        */
        private DataObject populateThisNode(DataObject inputObject, String entity,
            String key) throws RemoteException, ResourceException {
            getLogUtils().trace(Level.FINEST, RMRetrieveCommand.class.getName(),
                "populateThisNode()",
                "attempting to populate node: " + entity + " key=" + key);
            DataObject dataObject = null;
            RMDDataImplementation requestObject = new RMDDataImplementation();
            requestObject.setComponentName(entity);
            requestObject.setExecutionType("Retrieve");
            requestObject.setAttr("Id", key);
            RMDDataInterface responseObject = connection
                .retrieveObject(requestObject);

            if (responseObject == null) {
                getLogUtils().trace(Level.FINE, RMRetrieveCommand.class.getName(),
                    "populateThisNode()",
                    "Object not found " + entity + " " + key);
                throw new RecordNotFoundException(inputObject);
            } else {
                dataObject = super.getObjectConverter().toDataObject(
                    (RMDDataImplementation) responseObject, inputObject);
            }
            return dataObject;
        }
    }
}

```

11.6.10 Implement RMDeleteCommand class

This class implements business logic to delete recursively RedMaintenance records for a given parent key in the data object. Because only a parent key is given, in order to know if the parent has any off spring this class must rely on the application-specific information provided in the business object definition for parent child relationship information. For example, the Apartment business object definition must define that it can contain an array of Maintenance business objects. See the implementation in Example 11-13.

Example 11-13 RMDeleteCommand implementation

```

package com.ibm.itso.sab511.outbound.commandpattern;

import java.rmi.RemoteException;
import java.util.Iterator;
import java.util.List;
import java.util.Vector;
import java.util.logging.Level;
import javax.resource.ResourceException;

```

```

import javax.resource.spi.CommException;
import com.ibm.itso.rm.common.RMDataImplementation;
import com.ibm.itso.rm.common.RMDataInterface;
import com.ibm.itso.rm.common.RMServerRMIInterface;
import com.ibm.j2ca.extension.logging.LogUtilConstants;
import commonj.sdo.DataObject;

public class RMDeleteCommand extends RMBaseCommand {
    RMServerRMIInterface connection;

    public RMDeleteCommand() {
        super();
    }

    /**
     * @param inputObject
     */
    public RMDeleteCommand(commonj.sdo.DataObject inputObject) {
        super(inputObject);
    }

    /**
     * This method is called by Command Pattern's Interpreter to delete records
     * in RedMaintenance. If a node has children, the children are recursively
     * deleted from RedMaintenance.
     */
    public DataObject execute(DataObject inputRecord) throws ResourceException
    {
        getLogUtils().traceMethodEntrance(RMDeleteCommand.class.getName(),
            "execute()");
        String key = super.getObjectNaming().getKey(inputRecord);
        String entity = super.getObjectNaming().getEntityName(inputRecord);
        connection = (RMServerRMIInterface) this.getConnection();
        this.deleteOneRedMaintenanceRecord(entity, key);
        this.deleteChildren(inputRecord, entity, key);
        getLogUtils().traceMethodExit(RMDeleteCommand.class.getName(),
            "execute()");
        return inputRecord;
    }

    /**
     * Delete one record in RedMaintenance.
     *
     * @param entity -
     *             A RedMaintenance entity
     * @param key -
     *             A key for a RedMaintenance entity
     * @throws ResourceException
     */
}

```

```

private void deleteOneRedMaintenanceRecord(String entity, String key)
    throws ResourceException {
    getLogUtils().trace(Level.FINEST, RMRetrieveCommand.class.getName(),
        "deleteThisNode()",
        "attempting to delete node: " + entity + " key=" + key);
    try {
        RMDDataImplementation redMaintenanceObject = new
RMDDataImplementation();
        redMaintenanceObject.setComponentName(entity);
        redMaintenanceObject.setExecutionType("Delete");
        redMaintenanceObject.setAttr("Id", key);
        redMaintenanceObject.setAttr("Status", "D");
        connection.deleteObject(redMaintenanceObject);
    } catch (RemoteException e) {
        this.getLogUtils().log(Level.SEVERE,
            LogUtilConstants.ADAPTER_RBUNDLE,
            RMDeleteCommand.class.getName(), "execute()", "1003",
            new Object[] { key });
        throw new CommException(e);
    }
}

/**
 * Recursively delete children in RedMaintenance for a given parent data
 * object.
 *
 * @param parentDataObject
 * @param parentEntity
 * @param parentKey
 * @throws ResourceException
 */
private void deleteChildren(DataObject parentDataObject,
    String parentEntity, String parentKey) throws ResourceException {
    if (parentDataObject == null)
        return;
    getLogUtils().trace(
        Level.FINEST,
        RMRetrieveCommand.class.getName(),
        "deleteChildren()",
        "attempting to delete children in " + parentEntity + "."
        + parentKey);
    List childAttributes = findChildAttributes(parentDataObject);
    Iterator childAttributeIterator = childAttributes.iterator();
    while (childAttributeIterator.hasNext()) {
        String currentChildName = (String) childAttributeIterator.next();
        RMDDataImplementation requestObject = new RMDDataImplementation();
        requestObject.setComponentName(currentChildName);
        requestObject.setAttr(parentEntity + "Id", parentKey);
        try {

```

```

        RMDDataInterface responseObject = connection
            .retrieveObjectByValue(requestObject);
        Vector childObjects = responseObject.getRetrieveByValue();
        if (childObjects != null) {
            for (int i = 0; i < childObjects.size(); i++) {
                RMDDataInterface childInstance = (RMDDataInterface)
childObjects
                    .get(i);
                if (!childInstance.getAttr("Status").equals("D")) {
                    String childKey = childInstance.getAttr("Id");
                    DataObject childObject = parentDataObject
                        .createDataObject(currentChildName);
                    this.getObjectNaming()
                        .setKey(childObject, childKey);
                    this.deleteOneRedMaintenanceRecord(currentChildName,
                        childKey);
                    this.deleteChildren(childObject, currentChildName,
                        childKey);
                }
            }
        }
    } catch (NullPointerException e) {
        //DataObject has no children, recursion stops here.
        getLogUtils().trace(
            Level.FINE,
            RMRetrieveCommand.class.getName(),
            "deleteChildren()",
            super.getObjectNaming().getEntityName(parentDataObject)
                + " has no children");
    } catch (RemoteException e) {
        this.getLogUtils().log(Level.SEVERE,
            LogUtilConstants.ADAPTER_RBUNDLE,
            RMDeleteCommand.class.getName(), "deleteChildren()",
            "1003",
            new Object[] { parentKey });
        throw new CommException(e);
    }
}
}
}
}
}

```

11.6.11 Implement RMResourceAdapter

For outbound operations we implement the method `getResourceAdapterMetadata` in `RMResourceAdapter` class. We return a new instance of either `RMResourceAdapterMeta` or its parent `WBIRResourceAdapterMeta`. The `ResourceAdapterMeta` class contain

information such as the adapter name, adapter version and adapter vendor. We can specify this information in the constructor of the `WBIResourceAdapterMetadata`. For example, we can have the following return statement in the `getResourceAdapterMetadata` method:

```
return new WBIResourceAdapterMetadata("RedMaintenance","IBM","1.0",false);
```

Alternatively, we can specify this information in the file `versioninfo.jar.txt` under `Connector Projects/RedMaintenance/connectorModule/META-INF` directory. If we specify this information in `versioninfo.jar.txt`, we return a new instance of `WBIResourceAdapterMetadata` as follows in the `getResourceAdapterMetadata` method:

```
return new WBIResourceAdapterMetadata(this,false);
```

In our implementation we use the following steps to create `versioninfo.jar.txt`:

1. Switch to **J2EE** perspective.
2. Right-click **META-INF** directory under `Connector Projects/RedMaintenance/connectorModule` directory.
3. From the context menu, select **New**.
4. Select **Other**.
5. Select **Simple**.
6. Select **File**.
7. Click **Next**.
8. Enter `versioninfo.jar.txt` as the file name.
9. Click **Finish**.
10. Copy the content of Example 11-14 into this file.
11. **Save** this file.

Example 11-14 versioninfo.jar.txt file

```
Name: com/ibm/itso/sab511
Implementation-Title: "IBM WebSphere Adapter for RedMaintenance"
Implementation-Version: 1.0.0
Implementation-Vendor: "International Business Machines Corporation"
```

See Example 11-15 for a full implementation of `RMResourceAdapter` class.

Example 11-15 RMResourceAdapter implementation

```
package com.ibm.itso.sab511;

import com.ibm.j2ca.base.WBIPollableResourceAdapter;
import com.ibm.j2ca.base.WBIResourceAdapter;
```



```

import com.ibm.j2ca.base.WBIResourceAdapterMetadata;

public class RMResourceAdapter extends WBIResourceAdapter implements
    WBIPollableResourceAdapter {
    WBIResourceAdapterMetadata metadata = null;
    /**
     *
     */
    public RMResourceAdapter() {
        super();
        // TODO Auto-generated constructor stub
    }

    /** (non-Javadoc)
     * @see com.ibm.j2ca.base.WBIResourceAdapter#getResourceAdapterMetadata()
     */
    public com.ibm.j2ca.base.WBIResourceAdapterMetadata
    getResourceAdapterMetadata()
        throws javax.resource.ResourceException {
        if(metadata != null){
            return metadata;
        }else{
            return new WBIResourceAdapterMetadata(this,false);
        }
    }

    /** (non-Javadoc)
     * @see
    com.ibm.j2ca.base.WBIPollableResourceAdapter#createEventStore(javax.resource.sp
    i.ActivationSpec)
     */
    public com.ibm.j2ca.extension.eventmanagement.EventStore createEventStore(
        javax.resource.spi.ActivationSpec arg0)
        throws javax.resource.ResourceException {
        // TODO Auto-generated method stub
        return null;
    }
}

```

11.6.12 Implement outbound log messages

In order for our adapter to support internationalization, we do not hard code log messages in our Java classes. Instead we create log message files otherwise called resource bundles for our adapter. The resource bundle contains key and value pairs. See Example 11-16.

Example 11-16 Key and value pairs in log messages file

```
1001=RM0001E: Unable to connect to RedMaintenance Server: {0}.
1001.useraction= Check RedMaintenance Server is started.
1001.explanation= Unable to connect to RedMaintenance Server.
```

In our Java classes, we reference the message key when we need to output a log message. See Example 11-17.

Example 11-17 Reference the message key in a Java class

```
try {
    eisConnection = (RMServerRMIInterface) Naming.lookup(rmiURL);
    connected = true;
} catch (Exception e) {
    logUtils.log(Level.SEVERE, LogUtilConstants.ADAPTER_RBUNDLE,
        CLASS_NAME, "connectoToEIS", "1001", new Object[] {});
}
```

The base log message file and the translated log messages files are placed in the same package where our `RMResourceAdapter` class resides. At runtime, the foundation classes automatically resolves the log messages to the same language as the language and locale of the user's operating system. See Example 11-18 for outbound log messages.

Example 11-18 Outbound log messages in LogMessages.properties file

```
# -----
# This is the message file for the resource adapter
# (Component Prefix: XXXXX)
# -----
#
# Background:
# Each message in this file is comprised of three parts:
# [1] Message ID
#     should follow format NNNNNmmmmS where
#     NNNNN is a five letter component prefix
#     mmmm is the message number
#     S is a type identifier to identify the type of the message
#
#     the component prefix "XXXXX" is reserved for the family of WBI adapters
#     and adapter-related components.
#
# [2] Explanation
#     Provides a more in-depth explanation of the message; assumes that
#     the user may be unfamiliar with the entire meaning of the base message
#     itself.
#
# [3] User Action
```

```
#      If possible, one or more actions that the user can take to rectify
#      the situation, or to ensure that it doesn't happen again.
#
# SQLXCEPTION=XXXXX0001E: Wrong password.
# SQLXCEPTION.explanation=Wrong password.
# SQLXCEPTION.useraction=Change the password.

1001=RM0001E: Unable to connect to RedMaintenance Server: {0}.
1001.useraction= Check RedMaintenance Server is started.
1001.explanation= Unable to connect to RedMaintenance Server.

1002=RM0002E: Unable to create the object.
1002.useraction= Please verify the status of the RedMaintenance Server
1002.explanation= The create operation failed.

1003=RM0003E: Unable to delete the object. Key: {0}
1003.useraction= Please verify that the object exists in the database
1003.explanation= The delete operation failed.

1004=RM0004E: Unable to update the object. Key: {0}
1004.useraction= Please verify that the object exists in the database
1004.explanation= The update operation failed.

1005=RM0005E: Unable to retrieve the object. Key: {0}
1005.useraction= Please verify that the object exists in the database
1005.explanation= The retrieve operation failed.
```

11.6.13 Update MANIFEST.MF file

In your J2EE perspective, add the following content to the Connector Projects/RedMaintenance/connectorModule/META-INF/MANIFEST.MF file.

```
Manifest-Version: 1.0
Name: com/ibm/itso/sab511
Implementation-Title: "RedMaintenance Adapter"
Implementation-Version: 1.0.0
```

The information in this file is needed by Adapter Foundation Classes.

11.7 Create temporary SCA artifacts

In order to test our custom adapter on WebSphere Process Server, we need to have the following SCA artifacts. Most of these artifacts can be generated

automatically by Enterprise Service Discovery (ESD). Because we have not yet implemented RedMaintenance EMD component, we create them manually.

- ▶ Apartment, Maintenance, and PartOrder business objects
These business objects contain application-specific information that our custom adapter needs to process the business objects.
- ▶ RedMaintenance ASI schema
This schema constrains the application-specific business information annotation tags in the business object definitions.
- ▶ discovery-service.xml file
This file specifies among other things the namespace and location of RedMaintenance ASI schema.
- ▶ RedMaintenanceOutboundInterface.import file
This SCA EIS import binding. file binds the adapter to SCA in WebSphere Process Server.

Use WebSphere Integration Developer to generate the following:

- ▶ Apartment, Maintenance, and PartOrder business graphs
- ▶ RedMaintenanceOutboundInterface.wsdl interface file

Note: When we have developed Enterprise Metadata Discovery (EMD) for RedMaintenance, we can use ESD to automatically create SCA artifacts. EMD development can be complex. If you do not intend to develop EMD, this section shows you how to manually create the necessary SCA artifacts to test and run your custom adapter.

11.7.1 Create business objects with application-specific information

application-specific information (ASI) in business object definitions provides the adapter with application dependent instructions on how to process business objects. Corresponding ASI can be specified at the business object level, verb level, and attribute level of the business object definition. The adapter parses the ASI at runtime to generate queries for Create, Retrieve, Update and Delete operations. For example, we can use ASI to specify the Id attribute of Apartment business object as the primary key of the business object. During a retrieve operation, the adapter parses the incoming business object, finds the key of the business object from attribute ASI and uses this key to retrieve the corresponding record in RedMaintenance. See example of ASI for Id attribute of Apartment business object in Example 11-19 on page 299.

Example 11-19 Apartment's Id attribute with ASI

```
<element name="Id" type="int" minOccurs="1" maxOccurs="1">
  <annotation>
    <appinfo

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
  <asi:PropertyTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
    <asi:FieldName>Id</asi:FieldName>
    <asi:PrimaryKey>true</asi:PrimaryKey>
  </asi:PropertyTypeMetadata>
</appinfo>
</annotation>
</element>
```

To manually create a business object in WebSphere Integration Developer, perform the following steps. Many of the steps are similar to the ones we perform to create the sample HelloWorld adapter:

1. Open **Business Integration** perspective.
2. Create a new Module called RedMaintenanceModule.
3. Right-click **RedMaintenanceModule**.
4. Select **Open Dependency Editor**.
5. In the Dependency Editor, add **RedMaintenance connector project** to the J2EE dependency section.
6. On the menu bar select **Window** → **Show view** → **Physical Resources** to switch to **Physical Resources** view.
7. Create a new a simple file name Apartment.xsd under RedMaintenanceModule folder.
8. Copy the corresponding content from the examples below into this file.
9. **Save** this file.
10. Repeat the previous three steps to create **Maintenance.xsd** and **PartOrder.xsd**.

Note: You see warnings when you create these files. The warnings disappear after you create **RedMaintenanceASI.xsd**, **BaseAdapterMetadata.xsd**, and **discovery-service.xml** files. If the warnings do not go away, click **Project** → **Clean** to clean the project.

Apartment business object definition with ASI is shown in Example 11-20. Notice that it imports Maintenance business object definition. This is because Maintenance is a child object of this business object. It also imports RedMaintenanceASI schema. RedMaintenanceASI schema is used to constrain ASI tags in the business object.

Example 11-20 Apartment.xsd

```

<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/ap
artment"

xmlns:apartment="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/ap
artment"
    xmlns:rmASI="asi"

xmlns:maintenance="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/
maintenance">
    <import

namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/maintena
nce"
        schemaLocation="Maintenance.xsd" />
    <import

namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
"
        schemaLocation="RedMaintenanceASI.xsd" />

    <annotation>
        <appinfo source="commonj.connector.asi">
            <asi:annotationSet xmlns:asi="commonj.connector.asi"

asiNSURI="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata"
/>
                </appinfo>
            </annotation>
            <complexType name="Apartment">
                <annotation>
                    <appinfo

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
                        <asi:BusinessObjectTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
                                <asi:ObjectName>Apartment</asi:ObjectName>

```

```

        </asi:BusinessObjectTypeMetadata>
    </appinfo>
</annotation>
<sequence minOccurs="1" maxOccurs="1">
    <element name="Id" type="int" minOccurs="1" maxOccurs="1">
        <annotation>
            <appinfo
source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
                <asi:PropertyTypeMetadata
xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
                    <asi:FieldName>Id</asi:FieldName>
                    <asi:PrimaryKey>true</asi:PrimaryKey>
                </asi:PropertyTypeMetadata>
            </appinfo>
        </annotation>
    </element>
    <element name="ApartmentNumber" type="int" minOccurs="0"
maxOccurs="1">
        <annotation>
            <appinfo
source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
                <asi:PropertyTypeMetadata
xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
                    <asi:FieldName>
                        ApartmentNumber
                    </asi:FieldName>
                </asi:PropertyTypeMetadata>
            </appinfo>
        </annotation>
    </element>
    <element name="Status" type="string" minOccurs="0"
maxOccurs="1">
        <annotation>
            <appinfo
source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
                <asi:PropertyTypeMetadata
xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
                    <asi:FieldName>Status</asi:FieldName>
                </asi:PropertyTypeMetadata>
            </appinfo>

```

```

        </annotation>
    </element>
    <element name="AddressLine1" type="string" minOccurs="0"
        maxOccurs="1">
        <annotation>
            <appinfo
                source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
                <asi:PropertyTypeMetadata>

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
                    <asi:FieldName>AddressLine1</asi:FieldName>
                </asi:PropertyTypeMetadata>
            </appinfo>
        </annotation>
    </element>
    <element name="AddressLine2" type="string" minOccurs="0"
        maxOccurs="1">
        <annotation>
            <appinfo
                source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
                <asi:PropertyTypeMetadata>

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
                    <asi:FieldName>AddressLine2</asi:FieldName>
                </asi:PropertyTypeMetadata>
            </appinfo>
        </annotation>
    </element>
    <element name="AddressLine3" type="string" minOccurs="0"
        maxOccurs="1">
        <annotation>
            <appinfo
                source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
                <asi:PropertyTypeMetadata>

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
                    <asi:FieldName>AddressLine3</asi:FieldName>
                </asi:PropertyTypeMetadata>
            </appinfo>
        </annotation>
    </element>
    <element name="AddressLine4" type="string" minOccurs="0"
        maxOccurs="1">

```



```

        <annotation>
          <appinfo

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
          <asi:PropertyTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
            <asi:FieldName>AddressLine4</asi:FieldName>
          </asi:PropertyTypeMetadata>
        </appinfo>
      </annotation>
    </element>
    <element name="PostCode" type="string" minOccurs="0"
      maxOccurs="1">
      <annotation>
        <appinfo

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
        <asi:PropertyTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
          <asi:FieldName>PostCode</asi:FieldName>
        </asi:PropertyTypeMetadata>
      </appinfo>
    </annotation>
  </element>
  <element name="Maintenance" type="maintenance:Maintenance"
    minOccurs="0" maxOccurs="unbounded">
    <annotation>
      <appinfo

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
      <asi:PropertyTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
        <asi:FieldName>Maintenance</asi:FieldName>
      </asi:PropertyTypeMetadata>
    </appinfo>
  </annotation>
</element>
</sequence>
</complexType>
</schema>

```

The Maintenance business object definition with ASI is shown in Example 11-21.

Example 11-21 Maintenance.xsd

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/ma
intenance"

xmlns:maintenance="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/
maintenance"
    xmlns:rmASI="asi"

xmlns:partorder="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/pa
rtorder">
    <import

namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/partorde
r"
        schemaLocation="PartOrder.xsd" />
    <import

namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
"
        schemaLocation="RedMaintenanceASI.xsd" />

    <annotation>
        <appinfo source="commonj.connector.asi">
            <asi:annotationSet xmlns:asi="commonj.connector.asi"

asiNSURI="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata"
/>
                </appinfo>
            </annotation>
            <complexType name="Maintenance">
                <annotation>
                    <appinfo

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
                        <asi:BusinessObjectTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
                            <asi:ObjectName>Maintenance</asi:ObjectName>
                            </asi:BusinessObjectTypeMetadata>
                        </appinfo>
                    </annotation>
                    <sequence minOccurs="1" maxOccurs="1">
```

```

        <element name="Id" type="int" minOccurs="1" maxOccurs="1">
            <annotation>
                <appinfo
                    source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
                        <asi:PropertyTypeMetadata>

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
                            <asi:FieldName>Id</asi:FieldName>
                            <asi:PrimaryKey>true</asi:PrimaryKey>
                        </asi:PropertyTypeMetadata>
                    </appinfo>
                </annotation>
            </element>
            <element name="ApartmentId" type="int" minOccurs="1"
                maxOccurs="1">
                <annotation>
                    <appinfo
                        source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
                            <asi:PropertyTypeMetadata>

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
                                <asi:FieldName>ApartmentId</asi:FieldName>
                            </asi:PropertyTypeMetadata>
                        </appinfo>
                    </annotation>
                </element>
                <element name="TenantId" type="int" minOccurs="1"
                    maxOccurs="1">
                    <annotation>
                        <appinfo
                            source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
                                <asi:PropertyTypeMetadata>

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
                                    <asi:FieldName>TenantId</asi:FieldName>
                                </asi:PropertyTypeMetadata>
                            </appinfo>
                        </annotation>
                    </element>
                    <element name="Status" type="string" minOccurs="0"
                        maxOccurs="1">
                        <annotation>
                            <appinfo

```

```

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
    <asi:PropertyTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
        <asi:FieldName>Status</asi:FieldName>
    </asi:PropertyTypeMetadata>
</appinfo>
</annotation>
</element>
<element name="ProblemDescription" type="string"
    minOccurs="0" maxOccurs="1">
    <annotation>
        <appinfo

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
    <asi:PropertyTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
        <asi:FieldName>
            ProblemDescription
        </asi:FieldName>
    </asi:PropertyTypeMetadata>
</appinfo>
</annotation>
</element>
<element name="StatusDescription" type="string"
    minOccurs="0" maxOccurs="1">
    <annotation>
        <appinfo

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
    <asi:PropertyTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
        <asi:FieldName>
            StatusDescription
        </asi:FieldName>
    </asi:PropertyTypeMetadata>
</appinfo>
</annotation>
</element>
<element name="ExpectedCompletion" type="long" minOccurs="0"
    maxOccurs="1">
    <annotation>
        <appinfo

```

```

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
    <asi:PropertyTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
        <asi:FieldName>
            ExpectedCompletion
        </asi:FieldName>
    </asi:PropertyTypeMetadata>
</appinfo>
</annotation>
</element>
<element name="ActualCompletion" type="long" minOccurs="0"
maxOccurs="1">
    <annotation>
        <appinfo

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
    <asi:PropertyTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
        <asi:FieldName>
            ActualCompletion
        </asi:FieldName>
    </asi:PropertyTypeMetadata>
</appinfo>
</annotation>
</element>
<element name="PartOrder" type="partorder:PartOrder"
minOccurs="0" maxOccurs="unbounded">
    <annotation>
        <appinfo

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
    <asi:PropertyTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
        <asi:FieldName>PartOrder</asi:FieldName>
    </asi:PropertyTypeMetadata>
</appinfo>
</annotation>
</element>
</sequence>
</complexType>
</schema>

```

PartOrder business object with ASI is shown in Example 11-22.

Example 11-22 PartOrder.xsd

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/pa
rtorder"

xmlns:partorder="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/pa
rtorder"
  xmlns:rmASI="asi">
  <import

namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
"
  schemaLocation="RedMaintenanceASI.xsd" />

  <annotation>
    <appinfo source="commonj.connector.asi">
      <asi:annotationSet xmlns:asi="commonj.connector.asi"

asiNSURI="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata"
/>
      </appinfo>
    </annotation>
    <complexType name="PartOrder">
      <annotation>
        <appinfo

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
        <asi:BusinessObjectTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
          <asi:ObjectName>PartOrder</asi:ObjectName>
          </asi:BusinessObjectTypeMetadata>
        </appinfo>
      </annotation>
      <sequence minOccurs="1" maxOccurs="1">
        <element name="Id" type="int" minOccurs="1" maxOccurs="1">
          <annotation>
            <appinfo

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
            <asi:PropertyTypeMetadata
```

```

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
    <asi:FieldName>Id</asi:FieldName>
    <asi:PrimaryKey>true</asi:PrimaryKey>
    </asi:PropertyTypeMetadata>
    </appinfo>
    </annotation>
</element>
<element name="MaintenanceId" type="int" minOccurs="0"
maxOccurs="1">
    <annotation>
    <appinfo>

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
    <asi:PropertyTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
    <asi:FieldName>MaintenanceId</asi:FieldName>
    </asi:PropertyTypeMetadata>
    </appinfo>
    </annotation>
</element>
<element name="PartNumber" type="string" minOccurs="0"
maxOccurs="1">
    <annotation>
    <appinfo>

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
    <asi:PropertyTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
    <asi:FieldName>PartNumber</asi:FieldName>
    </asi:PropertyTypeMetadata>
    </appinfo>
    </annotation>
</element>
<element name="Status" type="string" minOccurs="0"
maxOccurs="1">
    <annotation>
    <appinfo>

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
    <asi:PropertyTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">

```

```

        <asi:FieldName>Status</asi:FieldName>
      </asi:PropertyTypeMetadata>
    </appinfo>
  </annotation>
</element>
<element name="ExpectedDelivery" type="long" minOccurs="0"
  maxOccurs="1">
  <annotation>
    <appinfo>

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
      <asi:PropertyTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
        <asi:FieldName>
          ExpectedDelivery
        </asi:FieldName>
      </asi:PropertyTypeMetadata>
    </appinfo>
  </annotation>
</element>
<element name="ActualDelivery" type="long" minOccurs="0"
  maxOccurs="1">
  <annotation>
    <appinfo>

source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
      <asi:PropertyTypeMetadata

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
        <asi:FieldName>
          ActualDelivery
        </asi:FieldName>
      </asi:PropertyTypeMetadata>
    </appinfo>
  </annotation>
</element>
</sequence>
</complexType>
</schema>

```

11.7.2 Creating RedMaintenance ASI schema

To constrain what type of ASI can be added to a business object, we create a XML schema to define what ASI tags are allowed and how they are used in the business object definition. To facilitate reuse, we create two schemas:

► **BaseAdapterMetadata.xsd**

This schema specifies the generic constraints that can be used for other adapters.

► **RedMaintenanceASI.xsd**

This schema specifies the RedMaintenance specific constraints.

Use these steps to create these XML schemas:

1. Switch **J2EE** perspective.
2. Create these files under Connector **Projects** → **RedMaintenance** → **connectorModule** directory.
3. Copy and paste into the content of these files from Example 11-23 and Example 11-24 on page 312.
4. Click **File** → **Save**.

Both schemas in Example 11-23 and Example 11-24 on page 312 are almost the same. We create two schemas to demonstrate the concept of reusing a common schema.

Example 11-23 BaseAdapterMetadata.xsd

```
<xs:schema
targetNamespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/base/metadata"
xmlns:xs="http://www.w3.org/2001/XMLSchema"

xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/base/metadata"
elementFormDefault="qualified">

  <!-- ASI supported at BO-level in business object definitions -->
  <xs:complexType name="BusinessObjectTypeMetadata">
    <xs:sequence minOccurs="0" maxOccurs="1">
      <xs:element name="ObjectName" type="xs:string"/>
      <xs:element name="SupportedVerbs" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Create" type="asi:VerbTypeMetadata"/>
            <xs:element name="Update" type="asi:VerbTypeMetadata"/>
            <xs:element name="Delete" type="asi:VerbTypeMetadata"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

```

        </xs:element>
    </xs:sequence>
</xs:complexType>

<!-- ASI supported per top-level verb -->
<xs:complexType name="VerbTypeMetadata">
    <xs:sequence minOccurs="0" maxOccurs="1">
        <xs:element name="MethodName" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<!-- ASI supported at property-level in business object definitions -->
<xs:complexType name="PropertyTypeMetadata">
    <xs:sequence minOccurs="0" maxOccurs="1">
        <xs:element name="PrimaryKey" type="xs:boolean"/>
        <xs:element name="FieldName" type="xs:string"/>
        <xs:element name="FieldType" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
</xs:schema>

```

See Example 11-24.

Example 11-24 RedMaintenanceASL.xsd

```

<xs:schema

    targetNamespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/me
tadata"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"

    xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
"
    elementFormDefault="qualified">
    <xs:import
        namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/base/metadata"
        schemaLocation="BaseAdapterMetadata.xsd" />

    <!-- ASI supported at B0-level in business object definitions -->
    <xs:complexType name="BusinessObjectTypeMetadata">
        <xs:sequence minOccurs="0" maxOccurs="1">
            <xs:element name="ObjectName" type="xs:string" />
            <xs:element name="SupportedVerbs" minOccurs="0"
                maxOccurs="1">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Create"
                            type="asi:VerbTypeMetadata" />
                        <xs:element name="Update"

```

```

        type="asi:VerbTypeMetadata" />
        <xs:element name="Delete"
            type="asi:VerbTypeMetadata" />
        <xs:element name="Retrieve"
            type="asi:VerbTypeMetadata" />
    </xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>

<!-- ASI supported per top-level verb -->
<xs:complexType name="VerbTypeMetadata">
    <xs:sequence minOccurs="0" maxOccurs="1">
        <xs:element name="MethodName" type="xs:string" />
    </xs:sequence>
</xs:complexType>

<!-- ASI supported at property-level in business object definitions -->
<xs:complexType name="PropertyTypeMetadata">
    <xs:sequence minOccurs="0" maxOccurs="1">
        <xs:element name="PrimaryKey" type="xs:boolean" />
        <xs:element name="FieldName" type="xs:string" />
        <xs:element name="FieldType" type="xs:string" />
    </xs:sequence>
</xs:complexType>
</xs:schema>

```

11.7.3 Create discovery-service.xml file

This file specifies among other things the namespace and location of RedMaintenance ASI schema file to WebSphere Integration Developer.

Use these steps to create this file:

1. Switch to **J2EE** perspective.
2. Create a file name `discovery-service.xml` under Connector **Projects** → **RedMaintenance** → **connectorModule** → **META-INF** directory.
3. Copy the content of Example 11-25 on page 313 and paste into this file.
4. Click **File** → **Save**.

Example 11-25 discovery-service.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<emd:discoveryService xmlns:emd="commonj.connector"
    xmlns:j2ee="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

```

```

<j2ee:description>RedMaintenance EMD Adapter</j2ee:description>
<j2ee:display-name>RedMaintenance EMD Adapter</j2ee:display-name>
<emd:vendor-name xsi:type="j2ee:xsdStringType">IBM</emd:vendor-name>
<emd:version xsi:type="j2ee:xsdStringType">1.0.0</emd:version>
<emd:spec-version>1.0</emd:spec-version>
<emd:discoveryService-class
  xsi:type="j2ee:fully-qualified-classType">
  com.ibm.itso.sab511.emd.RMMetadataDiscovery
</emd:discoveryService-class>
<emd:metadataEdit-class xsi:type="j2ee:fully-qualified-classType">
  com.ibm.itso.sab511.emd.RMMetadataEdit
</emd:metadataEdit-class>
<emd:application-specific-schema>
  <j2ee:description>RedMaintenance ASI schema</j2ee:description>
  <j2ee:display-name>RedMaintenance ASI schema</j2ee:display-name>
  <emd:asiNSURI>
    http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
  </emd:asiNSURI>
  <emd:asiSchemaLocation>RedMaintenanceASI.xsd</emd:asiSchemaLocation>
</emd:application-specific-schema>
</emd:discoveryService>

```

11.7.4 Create Apartment business object graphs

Our custom adapter is designed to accept and return business graphs. Follow these steps to create an Apartment business graph:

1. Right-click **RedMaintenanceModule**.
2. Select **Open Dependency Editor**.
3. In the Dependency Editor, add **RedMaintenance connector project** to the J2EE dependency section.
4. On the menu bar, select **Window** → **Show view** → **Physical Resources** to switch to **Physical Resources** view.
5. Create a new a simple file name ApartmentBG.xsd under RedMaintenanceModule folder.
6. Copy the corresponding content from Example 11-26 into this file.
7. **Save** this file.

Example 11-26 ApartmentBG.xsd

```

<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/ap
artmentbg"

```

```

xmlns:apartment="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/ap
artment"
  xmlns:bo="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0">
  <import

namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/apartmen
t"
  schemaLocation="Apartment.xsd" />
  <import namespace="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0"
    schemaLocation="BusinessGraph.xsd" />
  <complexType name="ApartmentBG">
    <complexContent>
      <extension base="bo:BusinessGraph">
        <sequence>
          <element name="verb" minOccurs="0" maxOccurs="1">
            <simpleType>
              <restriction base="string">
                <enumeration value="Create" />
                <enumeration value="Update" />
                <enumeration value="Retrieve" />
                <enumeration value="Delete" />
              </restriction>
            </simpleType>
          </element>
          <element name="Apartment"
            type="apartment:Apartment" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</schema>

```

11.7.5 Create RedMaintenance outbound interface

Resource adapter declares its available operations in an SCA interface. Application service components call on the resource adapter by invoking the available operations in the adapter outbound interface. Creating an outbound interface for our RedMaintenance adapter is very similar to creating the HelloWorld outbound interface described in 8.4, “Create a Hello World adapter” on page 181. Follow these steps to create the RedMaintenanceOutboundInterface:

1. Switch to **Business Integration** perspective.
2. In the Business Integration view, right-click **Interface**.
3. In the context menu, select **New** → **Interface**.

4. Enter RedMaintenanceOutboundInterface in the name field of the New Interface Wizard.
5. In the interface editor create the operations for Apartment as shown in Figure 11-7 on page 316.

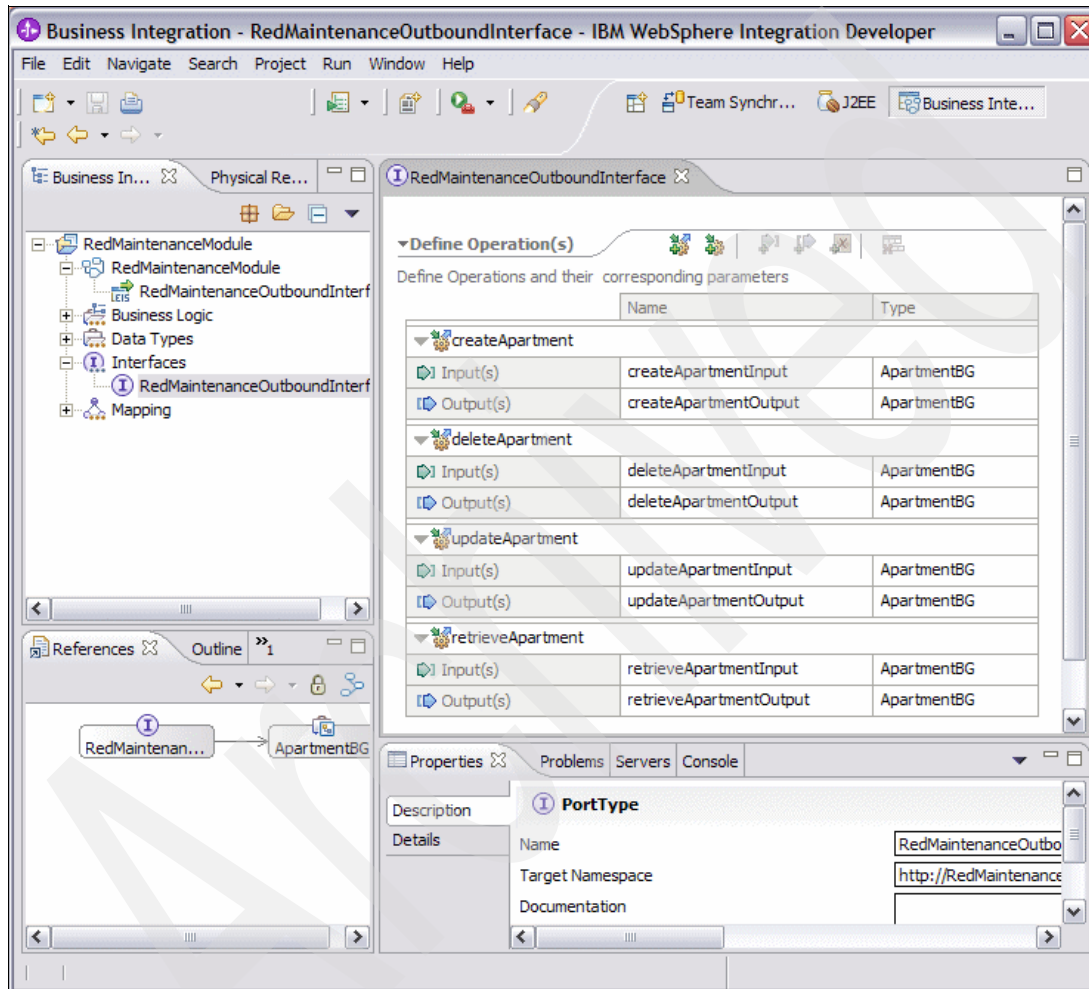


Figure 11-7 Apartment interface

Note: For detailed steps to create operations in an interface, see HelloWorld sample adapter in 8.4, “Create a Hello World adapter” on page 181.

6. Click **File** → **Save**.

11.7.6 Create SCA EIS service import file

The SCA EIS service import file specifies the mapping between the definition of outbound SCA interface, methods and data with the resource adapter.

1. Switch to **Business Integration** perspective.
1. On the menu bar select **Window** → **Show view** → **Physical Resources** to switch to **Physical Resources** view.
2. Create a new a simple file name `RedMaintenanceOutboundInterface.import` file under `RedMaintenanceModule` folder.
3. Copy the content from Example 11-27 into this file.
4. **Save** this file.

Example 11-27 RedMaintenanceOutboundInterface.import

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:import xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:eis="http://www.ibm.com/xmlns/prod/websphere/scdl/eis/6.0.0"
xmlns:ns1="http://RedMaintenanceModule/RedMaintenanceOutboundInterface"
xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/6.0.0"
xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/6.0.0"
displayName="RedMaintenanceOutboundInterface"
name="RedMaintenanceOutboundInterface">
  <interfaces>
    <interface xsi:type="wsdl:WSDLPortType"
portType="ns1:RedMaintenanceOutboundInterface">
      <method name="createApartment"/>
    </interface>
  </interfaces>
  <esbBinding xsi:type="eis:EISImportBinding"
dataBindingType="com.ibm.j2ca.extension.emd.runtime.WBIDataBindingImpl">
    <resourceAdapter name="RedMaintenanceModuleApp.RedMaintenance"
type="com.ibm.itso.sab511.RMResourceAdapter"/>
    <connection type="com.ibm.itso.sab511.outbound.RMManagedConnectionFactory"
interactionType="com.ibm.itso.sab511.outbound.RMInteractionSpec">
      <authentication resAuthAlias="widNode/Red_Alias"/>
    </connection>
    <methodBinding method="createApartment">
      <interaction>
        <properties>
          <functionName>Create</functionName>
        </properties>
      </interaction>
    </methodBinding>
    <methodBinding method="deleteApartment">
```

```

    <interaction>
      <properties>
        <functionName>Delete</functionName>
      </properties>
    </interaction>
  </methodBinding>
  <methodBinding method="updateApartment">
    <interaction>
      <properties>
        <functionName>Update</functionName>
      </properties>
    </interaction>
  </methodBinding>
  <methodBinding method="retrieveApartment">
    <interaction>
      <properties>
        <functionName>Retrieve</functionName>
      </properties>
    </interaction>
  </methodBinding>
</esbBinding>
</scdl:import>

```

11.7.7 Exporting the adapter

From our development environment, we can export our adapter in the following ways:

Tip: In some cases, changes to a business object definition might not show up in the exported EAR file. To ensure the exported EAR file contains the latest business object changes, close then reopen WebSphere Integration Developer before exporting the application EAR file.

- ▶ Export RedMaintenance module application .ear file that includes:
 - RedMaintenance adapter:
 - RedMaintenance adapter files
 - RedMaintenance common.jar file
 - WebSphere Adapter Foundation Classes
CWYBS_AdapterFoundation.jar file
 - Sample business objects definitions

- SCA outbound interface and EIS outbound binding

We can deploy this .ear file to WebSphere Process Server to test our adapter with the sample business objects and interface.

- Export RedMaintenance adapter .rar file that includes:

RedMaintenance adapter:

- RedMaintenance adapter files
- RedMaintenance common.jar file
- WebSphere Adapter Foundation Classes CWYBS_AdapterFoundation.jar file

We can use this adapter .rar file in new integration modules to create new integration applications.

- Export project files as a ProjectInterchange file to share our work with other developers. The following folders can be included in a ProjectInterchange file:
 - RedMaintenance connector project
 - RedMaintenanceModule integration module
 - RedMaintenanceModuleApp application
 - RedMaintenanceEJB EJB
 - RedMaintenanceEJBClient. EJB client
 - RedMaintenanceWeb Web application

Export RedMaintenance module application

Follow these steps to export RedMaintenance module application .ear file:

1. Switch to **J2EE** perspective.
2. Open the deployment descriptor under **Enterprise Applications** → **RedMaintenanceModuleApp** folder.
3. Select **Module**.
4. If you do not see Connector RedMaintenance.rar, click **Add** to add RedMaintenance.rar.
5. Right-click **Enterprise Applications** → **RedMaintenanceModuleApp** folder.
6. From the context menu, select **Export** → **EAR file**.
7. Enter a destination for the .ear file in the Export dialog box.
8. Click **Finish**.

Export RedMaintenance adapter

Follow these steps to export RedMaintenance adapter:

1. Switch to **J2EE** perspective.
2. Right-click **Connector Project** → **RedMaintenance**.

3. From the context menu, select **Export** → **RAR file**.
4. Enter a destination for the .rar file in the Export dialog box.
5. Click **Finish**.

Export RedMaintenance ProjectInterchange file

Follow these steps to export RedMaintenance ProjectInterchange file:

1. Click **File** → **Export**.
2. From the Export dialog box, select **ProjectInterchange**.
3. Click **Next**.
4. Select the folders you want to include in the ProjectInterchange file.
5. Enter a destination zip file name/path.
6. Click **Finish**.

11.8 Test outbound operations

In this section we explain how to test RedMaintenance's outbound operations. The following steps are performed for outbound testing:

1. Start **RedMaintenance server**.
2. Start **WebSphere Process Server** from WebSphere Integration Developer.
3. Install **RedMaintenance module** application to WebSphere Integration Developer's instance of WebSphere Process Server.
4. Test **createApartment** operation.
5. Test **retrieveApartment** operation.
6. Test **updateApartment** operation.
7. Test **deleteApartment** operation.

Start RedMaintenance server

If RedMaintenance server is not already started, follow these steps to start RedMaintenance:

1. From a command prompt, type `rmi registry` to start the **RMI registry**.
2. From your desktop, double-click **start_engine.bat** to start **RedMaintenance**.

Start WebSphere Process Server

If WebSphere Process Server is not already started, start **WebSphere Process Server** as shown in Figure 11-8.

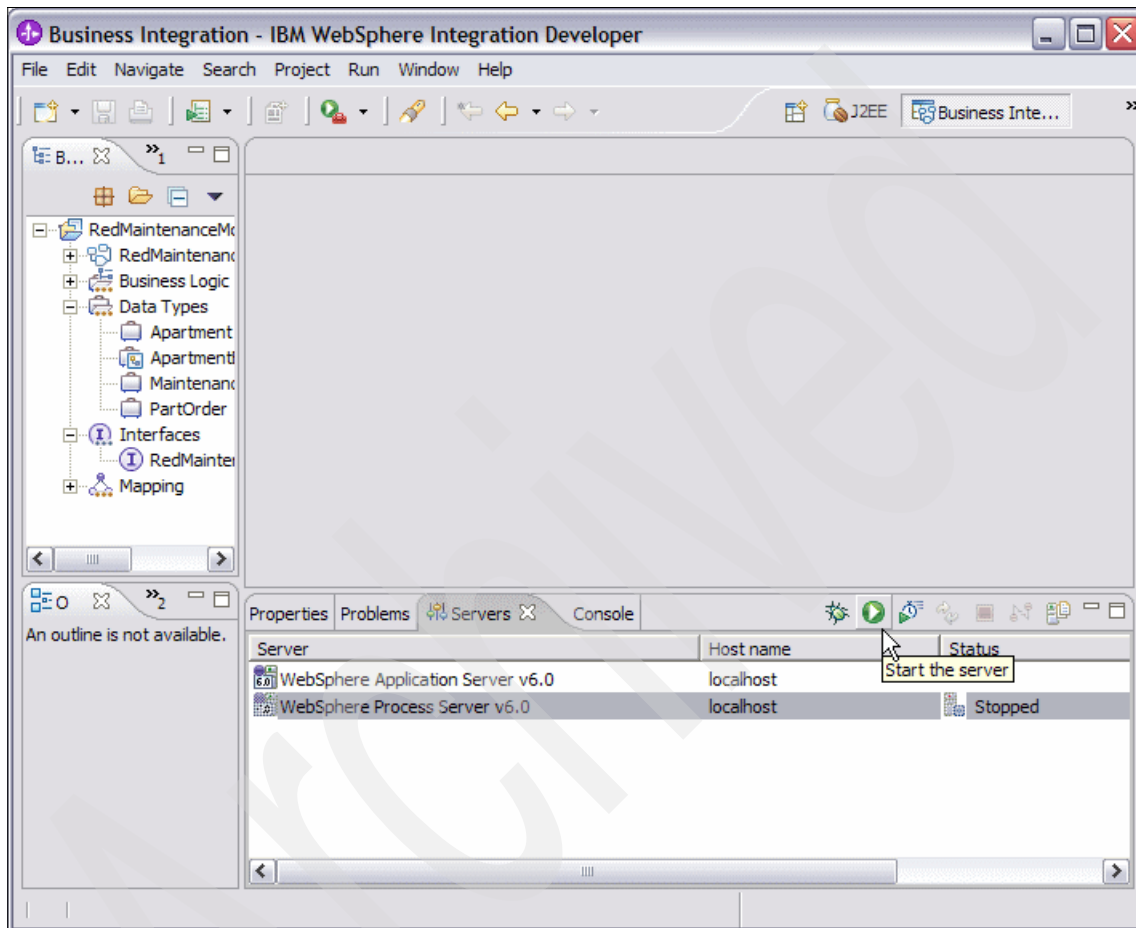


Figure 11-8 Start WebSphere Process Server

Configure server

Double-click **WebSphere Process Server v6.0** in the server view, then select **Run server with resource on Server**. This ensures that the test client we use for testing is using the adapter deployed on the server.

Install RedMaintenance module application

Installing RedMaintenance module application is very similar to installing Hello World application described in Chapter 8, "Setting up the development

environment” on page 159. Before installing RedMaintenance module application, we need to ensure the J2C authentication alias specified in RedMaintenanceOutboundInterface.import file exist in WebSphere Process Server’s J2C Authentication data configuration. You can verify this alias exist in RedMaintenanceOutboundInterface.import file either from the source xml file or from the properties view in WebSphere Integration Developer, see Figure 11-9.

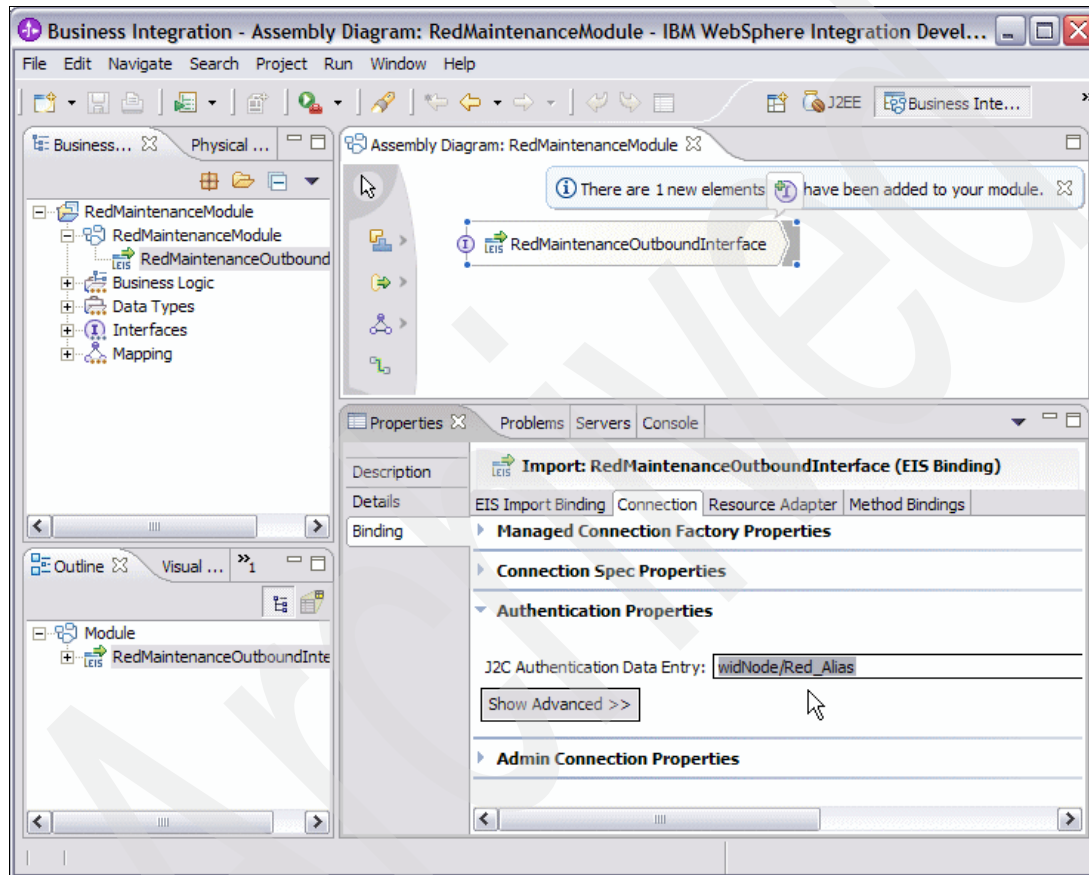


Figure 11-9 RedMaintenance outbound interface J2C authentication alias

From WebSphere Process Server's administrative console, you can verify this alias was created earlier in our Hello World adapter. See Figure 11-10.

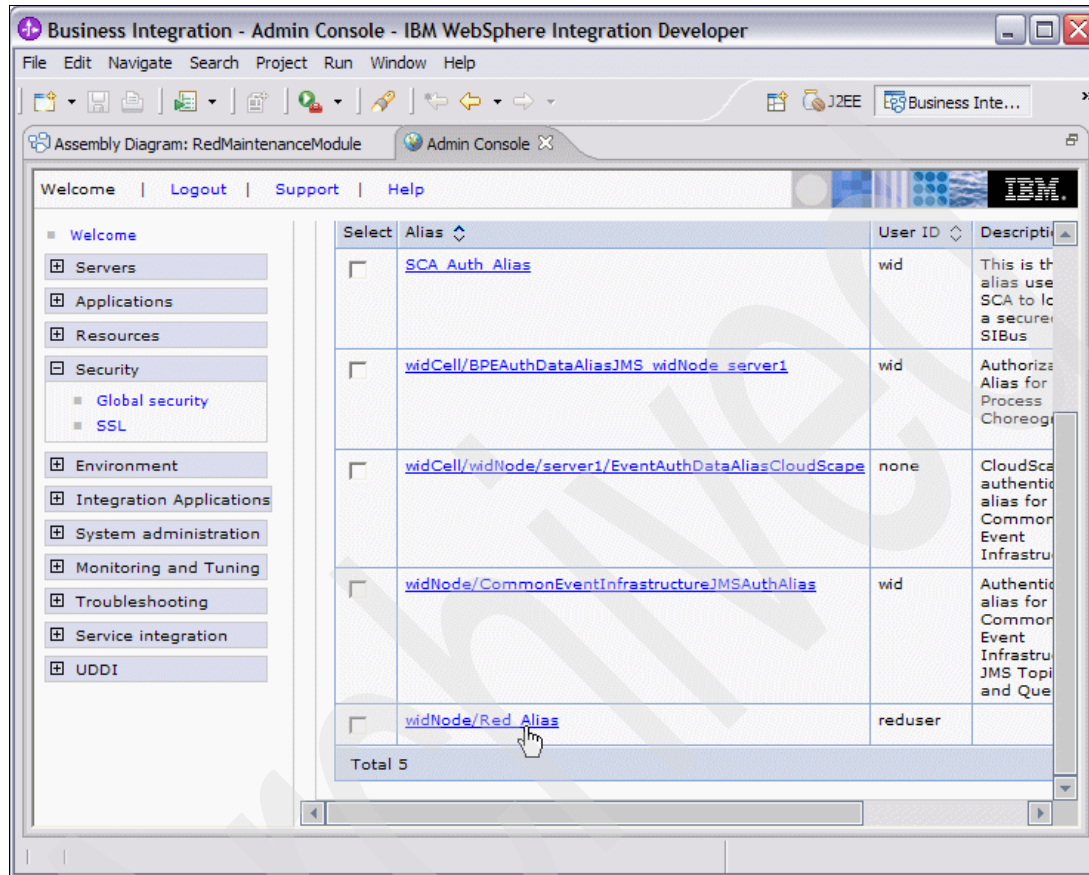


Figure 11-10 J2C authentication alias on WebSphere Process Server

If this alias does not exist on WebSphere Process Server, create this alias.

Note: RedMaintenance allows unauthenticated access to its API. Setting J2C authentication alias seems to be redundant. However, WebSphere Adapter Foundation classes that shipped with WebSphere Adapter Toolkit v. 6.0.0 require the setting of J2C authentication alias. This is not required in future versions of adapter foundation classes.

Follow these steps to install RedMaintenance module application to WebSphere Process Server:

1. In the server view, right-click **WebSphere Process Server v6.0**.
2. In the context menu, select **Run administrative console**.
3. Click **Login**.
4. Click the **+** beside **Applications** to expand it.
5. Click **Install New Application**.
6. Click **Browse** as shown in Figure 11-11 to browse to where you have exported the RedMaintenance module application.

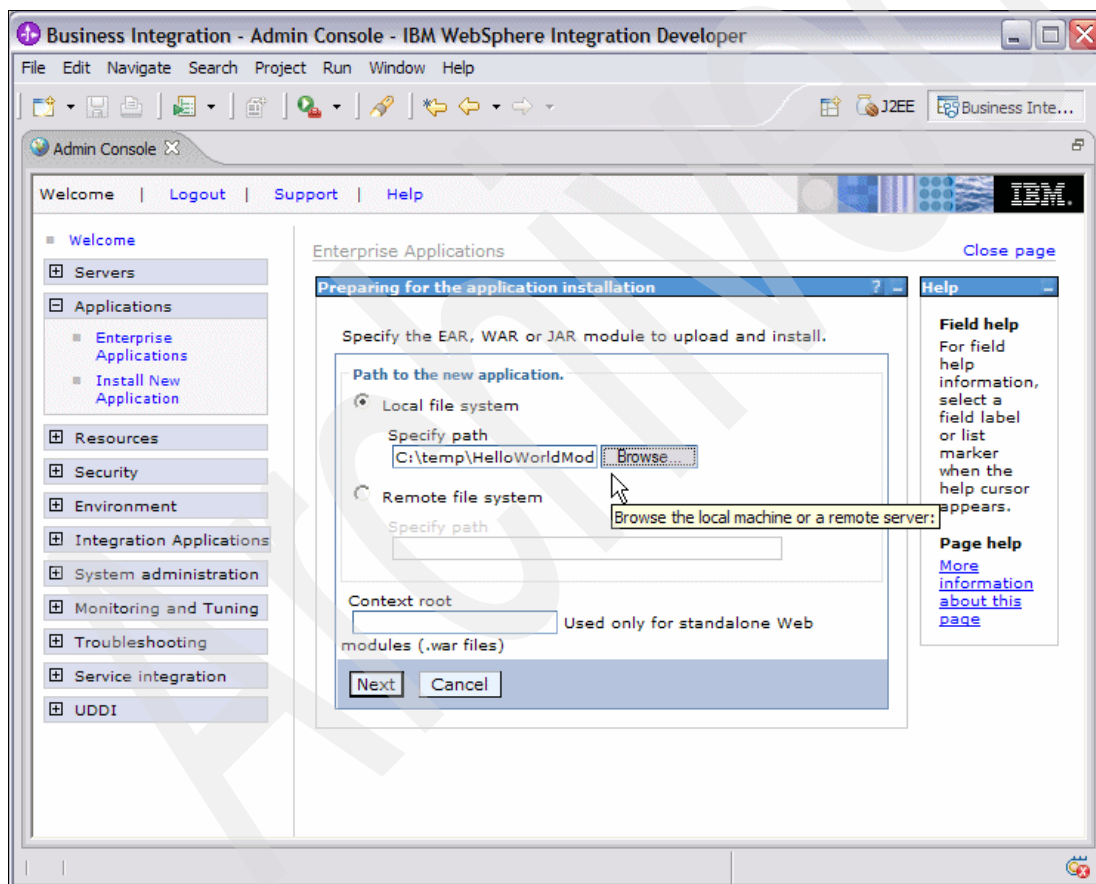


Figure 11-11 Install RedMaintenance module application

7. Click **Next**.
8. Click **Next**.

9. Click **Summary** (the last step).
10. Click **Finish**.
11. Click **Save to Master Configuration**.
12. Click **Save**.
13. On the left pane of the administrative console, click **Enterprise Applications** to show the installed applications.
14. Select **RedMaintenanceModuleApp**.
15. Select **Start** to start the application.

Test createApartment operation

In this test, we want to verify our adapter's ability to create entities in RedMaintenance. The created entity hierarchy must correspond to the input hierarchical Apartment business object. The input Apartment business object contains one Maintenance business object and the Maintenance business object contains one PartOrder business object.

Follow these steps to test the createApartment operation:

1. Switch to **Business Integration** perspective.
2. Open **Assembly Diagram** editor.
3. Right-click **RedMaintenanceOutboundInterface** import.
4. Select **Test Component**.
5. Select **Create** verb.
6. Select **Maintenance**.
7. Right-click **Maintenance**.
8. Select **Add Element** to add Maintenance properties.
9. Select **PartOrder**.
10. Right-click **PartOrder**.
11. Select **Add Element** to add PartOrder properties.

12. Enter sample values as shown in Figure 11-12.

Business Integration - RedMaintenanceOutboundInterface_Test - IBM WebSphere Integration Developer

File Edit Navigate Search Project Run Window Help

Admin Console Application Deployment Desc... Assembly Diagram: RedMaint... *RedMaintenanceOutboundIn...

Events

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

Events **General Properties** **Detailed Properties**

Invoke

Configuration: Default Module Test

Module: RedMaintenanceModule

Component: RedMaintenanceOutboundInterface

Interface: RedMaintenanceOutboundInterface

Operation: createApartment

Initial request parameters

Name	Type	Value
createApartmentInput	ApartmentBG	
verb	String	Create
Apartment	Apartment	
Id	int	0
Status	string	
ApartmentNumber	int	123
AddressLine1	string	456 Adapter Street
AddressLine2	string	Raleigh
AddressLine3	string	North Carolina
AddressLine4	string	USA
PostCode	string	12345
Maintenance	Maintenance []	
Maintenance[0]	Maintenance	
Id	int	0
ApartmentId	int	0
Status	string	
TenantId	int	0
ProblemDescription	string	Broken window
StatusDescription	string	
ExpectedCompletion	long	0
ActualCompletion	long	0
PartOrder	PartOrder []	
PartOrder[0]	PartOrder	
Id	int	0
Status	string	
MaintenanceId	int	0
PartNumber	string	9999
ExpectedDelivery	long	0
ActualDelivery	long	0

Data Pool Continue

Events Configurations

Figure 11-12 createApartment outbound operation

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

14. Examine the result of the createApartment operation. See Figure 11-13.

The screenshot shows the IBM WebSphere Integration Developer interface. The title bar reads "Business Integration - RedMaintenanceOutboundInterface_Test - IBM WebSphere Integration Developer". The menu bar includes File, Edit, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations and development tools. The main window is divided into several panes. On the left, the "Events" pane shows a tree view of the test execution: "Invoke (RedMaintenanceOutboundInterface)" is expanded, showing "Started", "Invoke (RedMaintenanceOutboundInterface)", "Return (RedMaintenanceOutboundInterface)", and "Stopped". The "General Properties" pane on the right shows the following details:

- Module: [RedMaintenanceModule](#)
- Component: [RedMaintenanceOutboundInterface](#)
- Interface: [RedMaintenanceOutboundInterface](#)
- Operation: [createApartment](#)

Below the properties, the "Return parameters:" section contains a table with the following data:

Name	Type	Value
createApartmentOutput	ApartmentBG	
verb	VerbType	Create
Apartment	Apartment	
Id	Integer	5
Status	String	
ApartmentNumber	Integer	123
AddressLine1	String	456 Adapter Street
AddressLine2	String	Raleigh
AddressLine3	String	North Carolina
AddressLine4	String	USA
PostCode	String	12345
Maintenance	Maintenance []	
Maintenance[0]	Maintenance	
Id	Integer	6
ApartmentId	Integer	5
Status	String	
TenantId	Integer	0
ProblemDescription	String	Broken window
StatusDescription	String	
ExpectedCompletion	Long	0
ActualCompletion	Long	0
PartOrder	PartOrder []	
PartOrder[0]	PartOrder	
Id	Integer	2
Status	String	
MaintenanceId	Integer	6
PartNumber	String	9999
ExpectedDelivery	Long	0
ActualDelivery	Long	0

At the bottom of the window, there are tabs for "Events" and "Configurations".

Figure 11-13 Result of createApartment operation

From the result of the createApartment operation, note that a new apartment entity, a new Maintenance entity, and a new PartOrder entity were created in RedMaintenance. The keys of the new entities were returned. Also note that the child business object contains a foreign key that points to the parent.

Test retrieveApartment operation

In this test we want to verify our adapter's ability to retrieve an entity in RedMaintenance. The retrieved entity must include all child entities if any from RedMaintenance.

Follow these steps to test retrieveApartment operation:

1. Switch to **Business Integration** perspective.
2. Open **Assembly Diagram** editor.
3. Right-click **RedMaintenanceOutboundInterface** import.
4. Select **Test Component**.
5. Select **retrieveApartment** operation as shown in Figure 11-14.

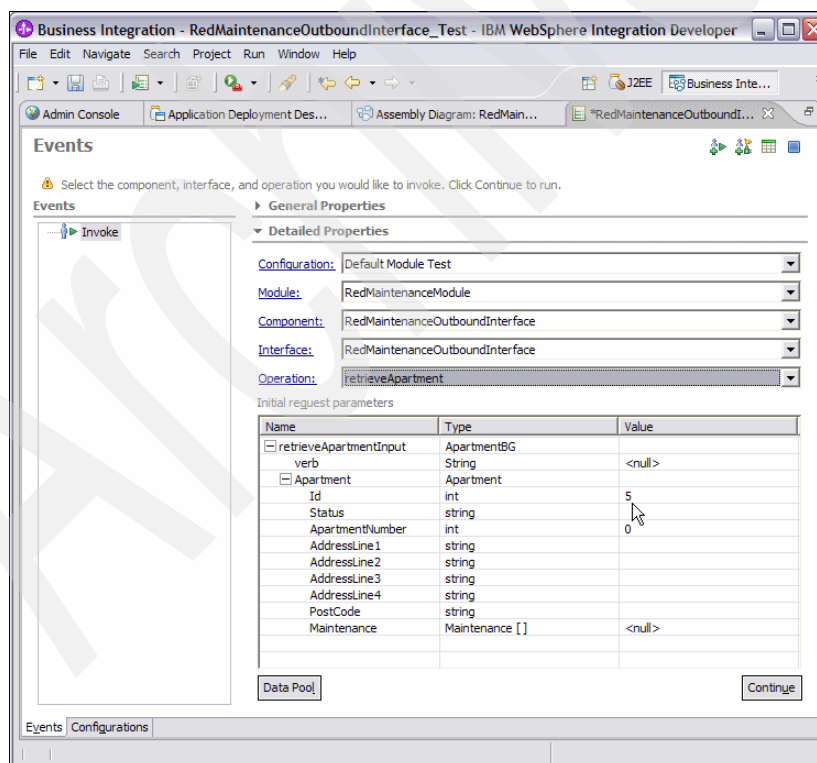


Figure 11-14 test retrieveApartment operation

6. Set Apartment key to the Apartment entity created in the createApartment operation test. See Figure 11-14 on page 328.
7. Click **Continue** to start the test.
8. Examine the result of the retrieveApartment operation. See Figure 11-15.

The screenshot shows the IBM WebSphere Integration Developer interface. The title bar reads "Business Integration - RedMaintenanceOutboundInterface_Test - IBM WebSphere Integration Developer". The main window is titled "Events" and contains two panes. The left pane shows a tree view of events: "Invoke (RedMaintenanceC)", "Started", "Invoke (RedMaintenanceC)", "Return (RedMaintenanceC)", and "Stopped". The right pane, titled "General Properties" and "Detailed Properties", shows the configuration for the "retrieveApartment" operation. It lists the Module as "RedMaintenanceModule", Component as "RedMaintenanceOutboundInterface", Interface as "RedMaintenanceOutboundInterface", and Operation as "retrieveApartment". Below this is a table of return parameters.

Name	Type	Value
retrieveApartmentOutput	ApartmentBG	
verb	VerbType	<null>
Apartment	Apartment	
Id	Integer	5
Status	String	A
ApartmentNumber	Integer	123
AddressLine1	String	456 Adapter Street
AddressLine2	String	Raleigh
AddressLine3	String	North Carolina
AddressLine4	String	USA
PostCode	String	12345
Maintenance	Maintenance []	
Maintenance[0]	Maintenance	
Id	Integer	6
ApartmentId	Integer	5
Status	String	A
TenantId	Integer	0
ProblemDescription	String	Broken window
StatusDescription	String	
ExpectedCompletion	Long	1135659600000
ActualCompletion	Long	1135659600000
PartOrder	PartOrder []	
PartOrder[0]	PartOrder	
Id	Integer	2
Status	String	N
MaintenanceId	Integer	6
PartNumber	String	9999
ExpectedDelivery	Long	-68400000
ActualDelivery	Long	-68400000

Figure 11-15 Result of retrieveApartment operation

From the result of the `retrieveApartment`, note that the hierarchy of `Apartment` object was return instead of just a single level `Apartment` object.

Test `updateApartment` operation

In this test we want to verify our adapter's ability to update entities in `RedMaintenance`. The adapter compares the incoming business object with existing `RedMaintenance` entities. It then performs `Update`, `Create`, and `Delete` operations to `RedMaintenance` entities so that the resulting `Apartment` entity hierarchy in `RedMaintenance` matches the incoming `Apartment` business object hierarchy. For more information about the `Update` operation, see the explanation of the `Command Pattern`.

Follow these steps to test the `retrieveApartment` operation:

1. Switch to **Business Integration** perspective.
2. Open **Assembly Diagram** editor.
3. Right-click **RedMaintenanceOutboundInterface** import.
4. Select **Test Component**.
5. Change the verb to `update`.
6. Select **updateApartment** as the outbound operation.
7. Enter sample update values as shown in Figure 11-16 on page 331.

Note: The out going business object does not contain any `PartOrder` child business object. Performing this update, the adapter effectively performs a delete to remove any children `PartOrder` entities under the parent `Maintenance` entity.

See the sample values in Figure 11-16.

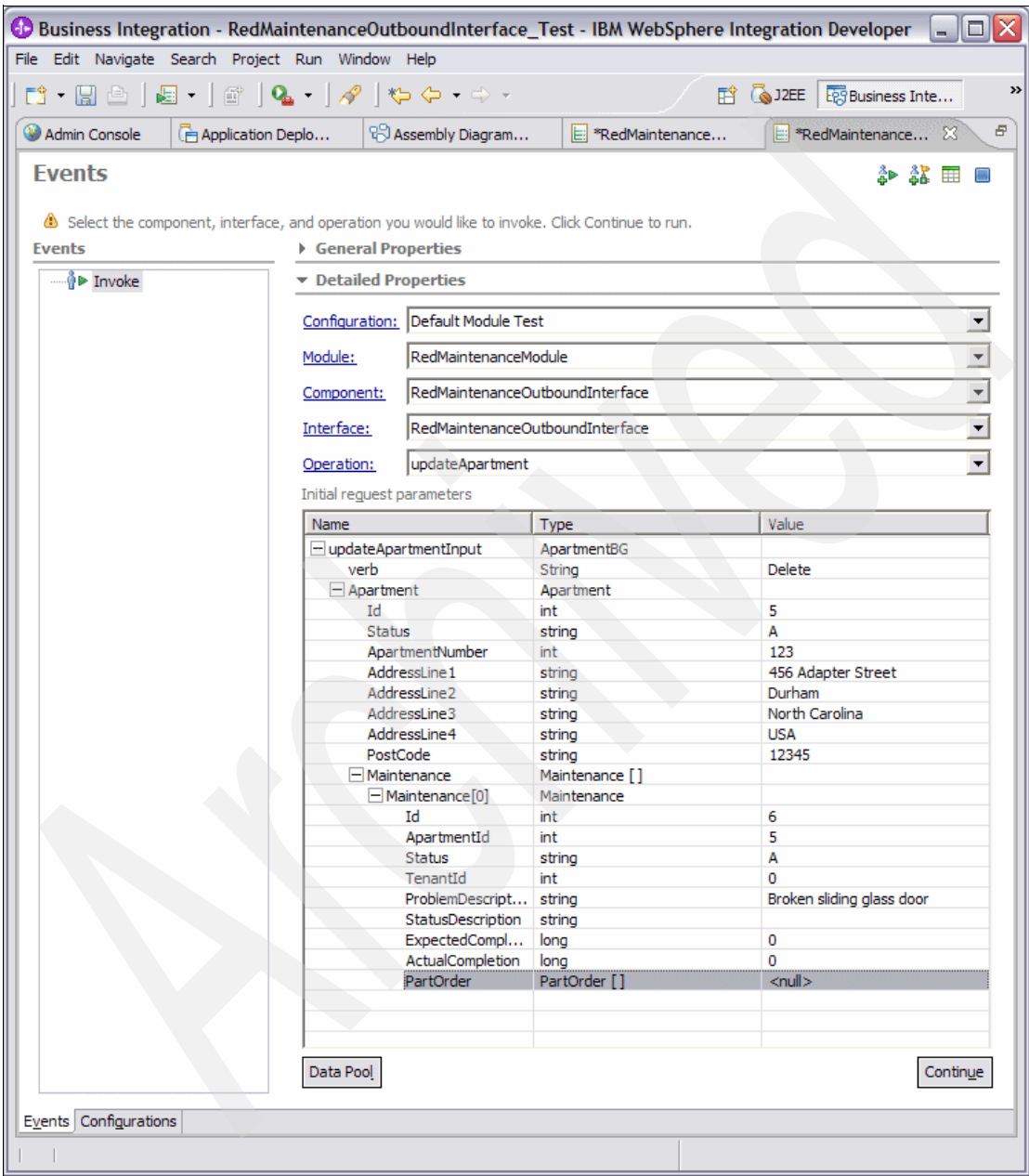


Figure 11-16 Test updateApartment operation

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

9. Examine the result of the updateApartment operation. See Figure 11-17.

The screenshot shows the IBM WebSphere Integration Developer interface. The title bar reads "Business Integration - RedMaintenanceOutboundInterface_Test - IBM WebSphere Integration Developer". The menu bar includes File, Edit, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations and development tools. The main workspace is divided into two panes. The left pane, titled "Events", shows a tree view of the event sequence: "Invoke (RedMaintenanceC)", "Started", "Invoke (RedMaintenance)", "Return (RedMaintenance)", and "Stopped". The right pane, titled "General Properties", shows the "Detailed Properties" for the selected event. It lists the Module as "RedMaintenanceModule", the Component as "RedMaintenanceOutboundInterface", the Interface as "RedMaintenanceOutboundInterface", and the Operation as "updateApartment". Below this, the "Return parameters:" section contains a table with the following data:

Name	Type	Value
updateApartmentOutput	ApartmentBG	
verb	VerbType	Update
Apartment	Apartment	
Id	Integer	5
Status	String	A
ApartmentNumber	Integer	123
AddressLine1	String	456 Adapter Street
AddressLine2	String	Durham
AddressLine3	String	North Carolina
AddressLine4	String	USA
PostCode	String	12345
Maintenance	Maintenance []	
Maintenance[0]	Maintenance	
Id	Integer	6
ApartmentId	Integer	5
Status	String	A
TenantId	Integer	0
ProblemDescript...	String	Broken sliding glass door
StatusDescription	String	
ExpectedCompl...	Long	0
ActualCompletion...	Long	0
PartOrder	PartOrder []	<null>

The bottom of the interface has tabs for "Events" and "Configurations".

Figure 11-17 Result of the updateApartment test

From the result of the `updateApartment`, note that the `Apartment` city was changed from Raleigh to Durham. Also note that the `PartOrder` child business object was deleted. For more information about the update operation see 4.11, “Command Pattern” on page 95.

Test `deleteApartment` operation

In this test we want to verify our adapter’s ability to delete entity in `RedMaintenance`. For a given parent, the adapter deletes the parent and all offspring entities from `RedMaintenance`.

Note: Delete in `RedMaintenance` are done using soft deletes.

Follow these steps to test the `retrieveApartment` operation:

1. Switch to **Business Integration** perspective.
2. Open **Assembly Diagram** editor.
3. Right-click **RedMaintenanceOutboundInterface** import.
4. Select **Test Component**.

5. Select **deleteApartment Operation** as shown in Figure 11-18.

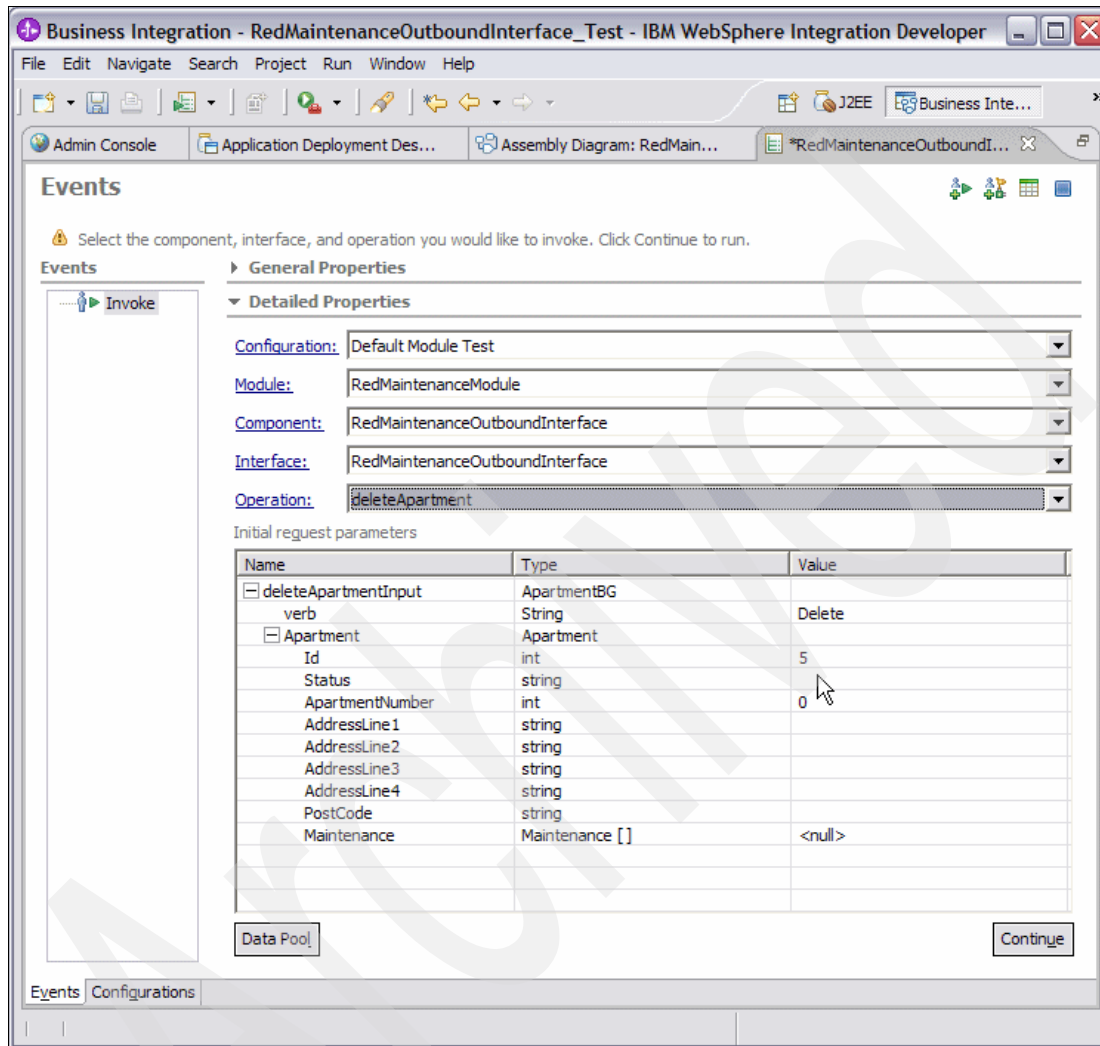
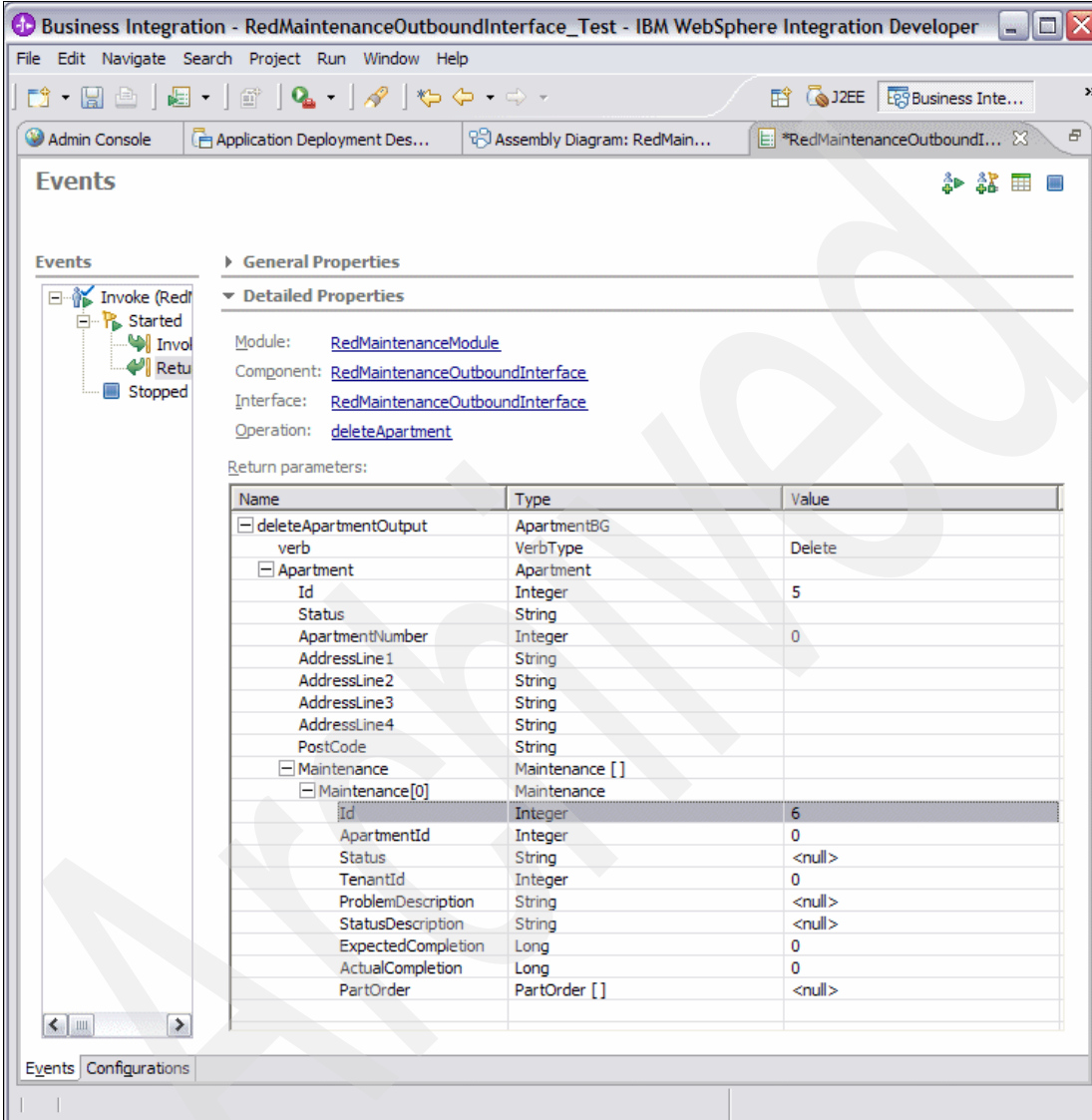


Figure 11-18 Test deleteApartment operation

6. Enter the Apartment id that was created in createApartment operation as shown in Figure 11-18.
7. Click **Continue** to start the test.

8. Examine the result of the deleteApartment operation. See Figure 11-19.



The screenshot shows the IBM WebSphere Integration Developer interface. The main window displays the 'Events' tab for the 'RedMaintenanceOutboundInterface_Test' project. The 'Events' list on the left shows the sequence: Invoke (RedMaintenanceModule), Started, Invoke (RedMaintenanceOutboundInterface), Return (RedMaintenanceOutboundInterface), and Stopped. The 'Detailed Properties' section shows the following information:

- Module: [RedMaintenanceModule](#)
- Component: [RedMaintenanceOutboundInterface](#)
- Interface: [RedMaintenanceOutboundInterface](#)
- Operation: [deleteApartment](#)

Return parameters:

Name	Type	Value
deleteApartmentOutput	ApartmentBG	
verb	VerbType	Delete
Apartment	Apartment	
Id	Integer	5
Status	String	
ApartmentNumber	Integer	0
AddressLine1	String	
AddressLine2	String	
AddressLine3	String	
AddressLine4	String	
PostCode	String	
Maintenance	Maintenance []	
Maintenance[0]	Maintenance	
Id	Integer	6
ApartmentId	Integer	0
Status	String	<null>
TenantId	Integer	0
ProblemDescription	String	<null>
StatusDescription	String	<null>
ExpectedCompletion	Long	0
ActualCompletion	Long	0
PartOrder	PartOrder []	<null>

Figure 11-19 Result of deleteApartment operation

From the result of the deleteApartment operation, note that the Apartment entity and its child Maintenance entity has been deleted.

11.9 Configure logging and tracing levels for debugging

In Chapter 7, “Exceptions, logging, and tracing” on page 147, we explain logging and tracing. Our code example shows you where you can place logging and trace code. Log levels allow you to control which events are processed by WebSphere Process Server’s logging mechanism. We can configure the runtime to output different levels of log and trace messages. The following steps illustrate how to configure the log and trace levels on WebSphere Process Server using the Admin Console:

1. Start **Admin Console** and select **Troubleshooting** as shown in Figure 11-20.

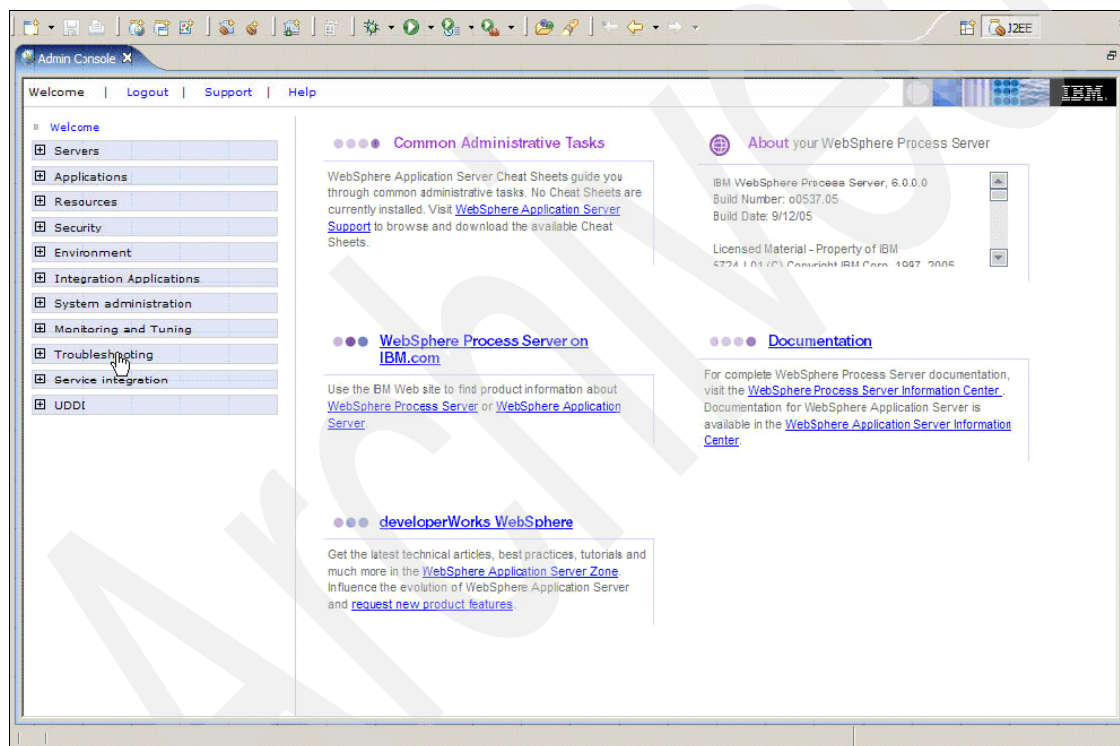


Figure 11-20 Troubleshooting

2. Select **Log** and **Trace** as shown in Figure 11-21.

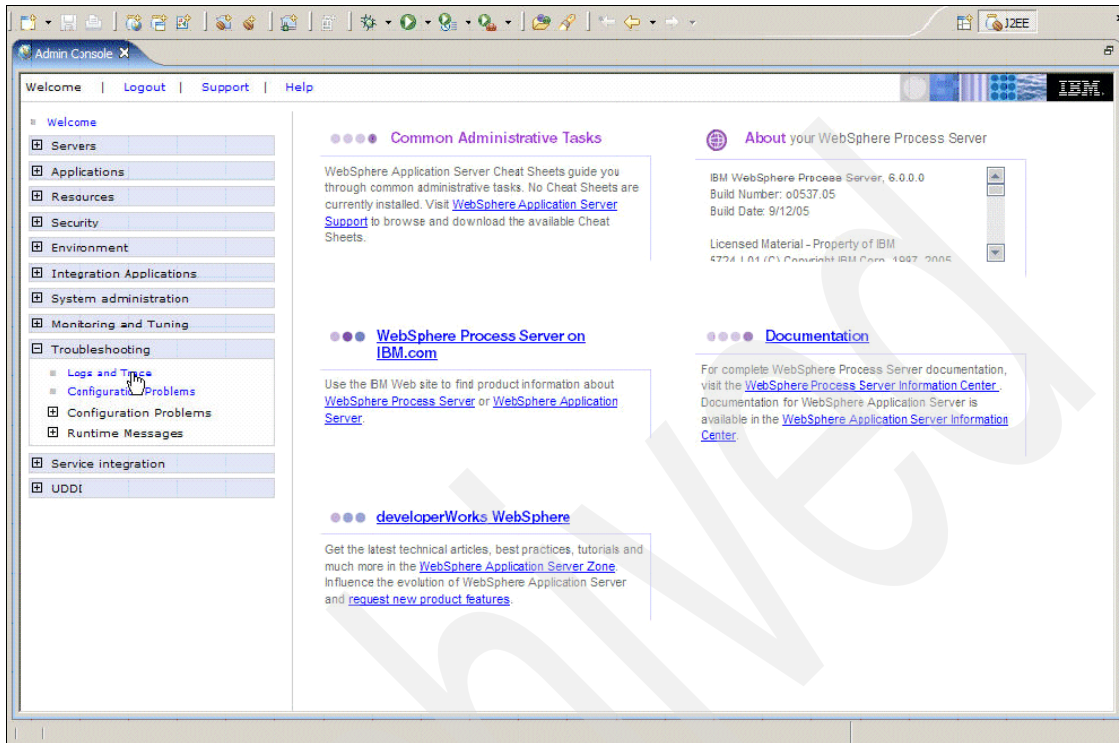


Figure 11-21 Log and Trace

3. Select your sever as shown in Figure 11-22.

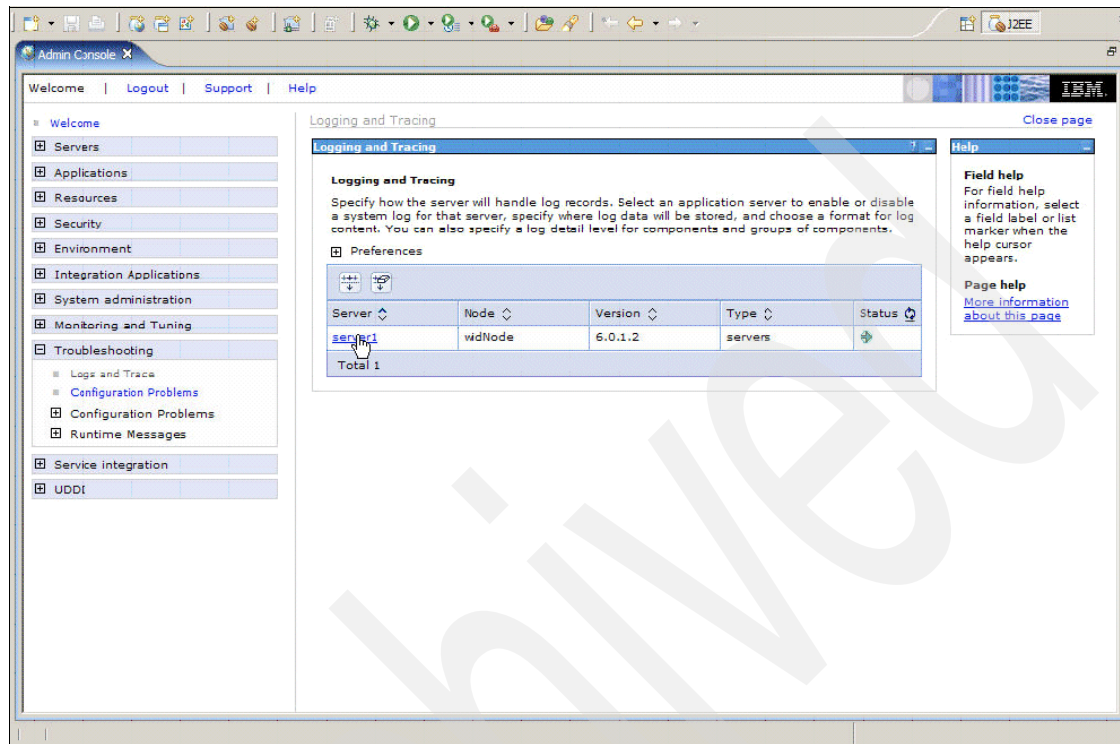


Figure 11-22 Server selection

4. Select **Change Log Detail Levels** as shown in Figure 11-23.

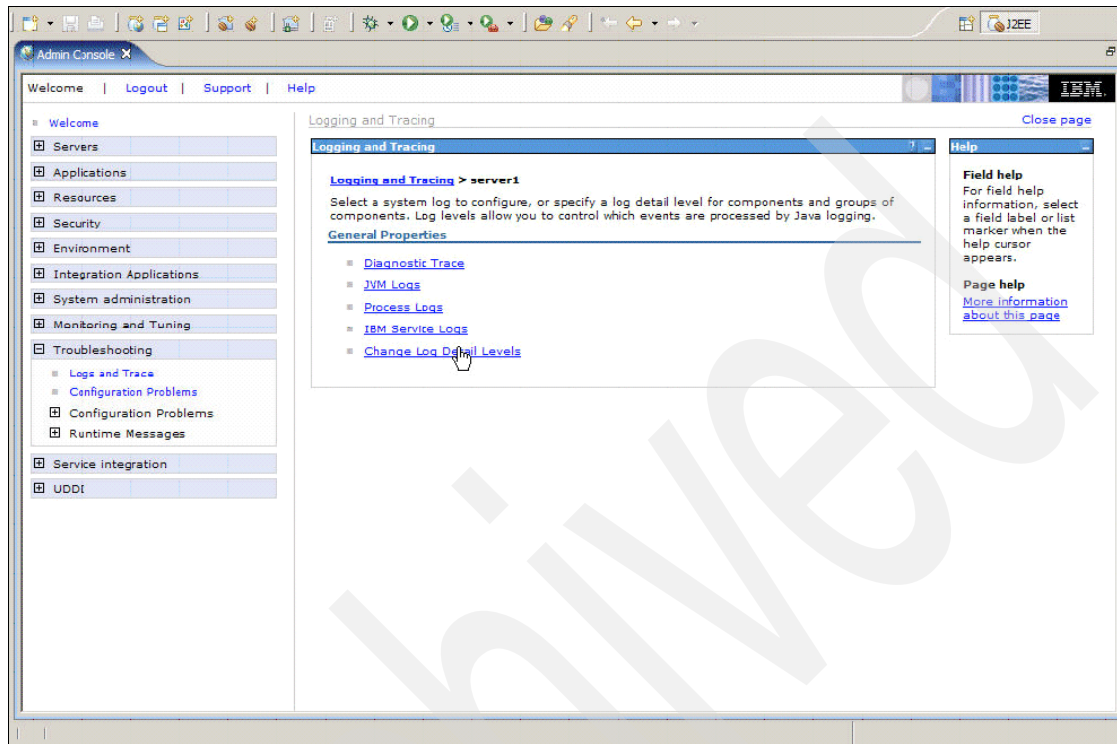


Figure 11-23 Change Log Detail Levels

5. Select the component that you want to configure. On the context menu, select your desired log and trace level. Figure 11-24 on page 340 shows different log and trace levels. Levels fatal to detail are logging levels while trace levels ranges from fine to finest. If you selected any of the trace levels, a trace file is produced. The default location of the trace file is located at `$(SERVER_LOG_ROOT)/trace.log`, where the variable `SERVER_LOG_ROOT` is the log directory for your server.

See Figure 11-24.

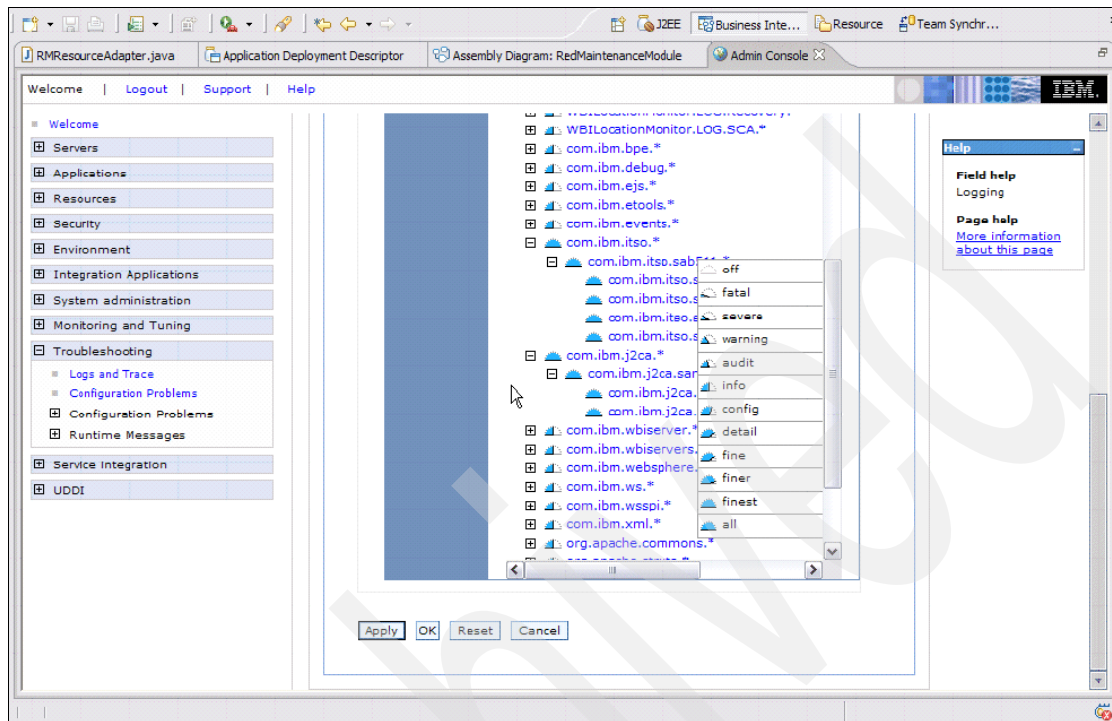


Figure 11-24 Set log and trace levels

6. Click **OK** and **Save** the changes.

After the appropriate log and trace level is set in the Admin Console, we can use our adapter to perform outbound or inbound operations (described in Chapter 12, “Implementing inbound processing and event handling” on page 343). We can then open the log or trace file to examine the events that are written to these files. See Example 11-28.

Example 11-28 Example trace.log file

```
[4/25/06 13:09:35:105 EDT] 0000004d ResourceAdapt 2 com.ibm.j2ca.base.WBIPollingTask run()
Entering method.
[4/25/06 13:09:35:105 EDT] 0000004d ResourceAdapt 2
com.ibm.j2ca.extension.eventmanagement.EventManager pollForEvents() Entering method.
[4/25/06 13:09:35:105 EDT] 0000004d ResourceAdapt 2
com.ibm.j2ca.extension.eventmanagement.SubscriptionManager recalculateSubscriptions() Entering
method.
```

```
[4/25/06 13:09:35:105 EDT] 0000004d ResourceAdapt <
com.ibm.j2ca.extension.eventmanagement.SubscriptionManager recalculateSubscriptions() Exiting
method.
[4/25/06 13:09:35:105 EDT] 0000004d ResourceAdapt <
com.ibm.j2ca.extension.eventmanagement.EndpointRecovery recoverInProgressEvents() Exiting
method.
[4/25/06 13:09:35:105 EDT] 0000004d ResourceAdapt 2 com.ibm.itso.sab511.inbound.RMEventStore
getEvents() pollQuantity: 10, status: 0
[4/25/06 13:09:35:105 EDT] 0000004d ResourceAdapt 2 com.ibm.itso.sab511.inbound.RMEventStore
getRMDDataImplementationInstance() Entering method.
[4/25/06 13:09:35:105 EDT] 0000004d ResourceAdapt < com.ibm.itso.sab511.inbound.RMEventStore
getRMDDataImplementationInstance() Exiting method.
[4/25/06 13:09:35:125 EDT] 0000004d ResourceAdapt 3 com.ibm.itso.sab511.inbound.RMEventStore
getEvents() No more events to process in EIS event table
[4/25/06 13:09:35:125 EDT] 0000004d ResourceAdapt 1 com.ibm.itso.sab511.inbound.RMEventStore
getEvents() Found 0 events to process
[4/25/06 13:09:35:125 EDT] 0000004d ResourceAdapt 2 com.ibm.itso.sab511.inbound.RMEventStore
getEvents() End processing: returnableList: 0
[4/25/06 13:09:35:125 EDT] 0000004d ResourceAdapt <
com.ibm.j2ca.extension.eventmanagement.EventManager pollForEvents() Exiting method.
[4/25/06 13:09:35:125 EDT] 0000004d ResourceAdapt < com.ibm.j2ca.base.WBIPollingTask run()
Exiting method.
```

Implementing inbound processing and event handling

This chapter explains how inbound processing for RedMaintenance adapter is implemented. Details on how to use WebSphere Adapter Foundation Classes library to speed up adapter inbound processing development are provided.

Adding inbound processing support to the adapter created in the previous involves the following:

- ▶ Generate inbound stub classes using WebSphere Adapter Toolkit.
- ▶ Implement inbound stub classes for RedMaintenance adapter.
- ▶ Create temporary SCA artifacts to test inbound processing.
- ▶ Test RedMaintenance adapter inbound processing.

12.1 Generate the inbound stub classes

In Chapter 11, “Implementing outbound request processing” on page 251, we explain how to use WebSphere Adapter Toolkit to create the stub classes for RedMaintenance adapter. We also explain how to implement the stub class to support adapter outbound processing. In this chapter we explain how to add inbound support for RedMaintenance adapter. If you are continuing from the previous chapter skip the next section and go directly to 12.1.1, “Start a new adapter project” on page 344. Otherwise you can use WebSphere Adapter Toolkit to create a new adapter project. This is described in the next section.

12.1.1 Start a new adapter project

Follow these steps to create a new adapter project using WebSphere Adapter Toolkit:

1. Launch the **New J2C Resource Adapter Project Wizard** to create a new resource adapter project and generate adapter stub classes:
 - a. Start **WebSphere Integration Developer**.
 - b. Select a new workspace when prompted.
 - c. WebSphere Integration Developer’s integrated development environment appears. Close the **Welcome** view to see the business integration perspective.
 - d. From the menu, select **File → New → Project**.
 - e. In the Select a wizard dialog box, select **Adapter Toolkit → J2C Resource Adapter Project**.
 - f. Click **Next**.
 - g. Enter RedMaintenance as the adapter project name in the Connector Project dialog.
 - h. Click **Next**.
 - i. In the J2C Resource Adapter Properties dialog, enter the following information in the Adapter Name, Package Name, Class Name Prefix fields:
 - RedMaintenance
 - com.ibm.itso.sab511
 - RM
 - j. Click **Next**.
 - k. In the Generation Options dialog box, select **IBM WebSphere Resource Adapter**.

Note: We selected IBM WebSphere Resource Adapter type to leverage the services provided by WebSphere Adapter Foundation Classes. The foundation classes help us to speed up development for adapters that are targeted for WebSphere Process Server. For differences between IBM WebSphere adapter and plain JCA resource adapter see Chapter 2, “Adapter basics” on page 33.

- l. In the same dialog box, select **Generate Inbound Adapter Classes**.

Note: If your adapter supports inbound operation and Enterprise Metadata Discovery, you can select them here. If your adapter supports bidirectional languages, for example Hebrew and Arabic, you can select **Generate Bidi Support Classes**.

- m. Click **Finish** to start generating selected components.
- n. Click **Yes** if you are prompted to switch your perspective to J2EE.

12.1.2 Using an existing adapter project

If the adapter project has already been created, follow these steps to add the inbound stub classes:

1. Open the **RedMaintenance** project, and edit the Resource Adapter Deployment Descriptor using the Deployment Descriptor Editor tool.
2. In the Overview panel of the Deployment Descriptor, look for the **Component Addition** option and click **Add**.

Figure 12-1 gives an example of that action.

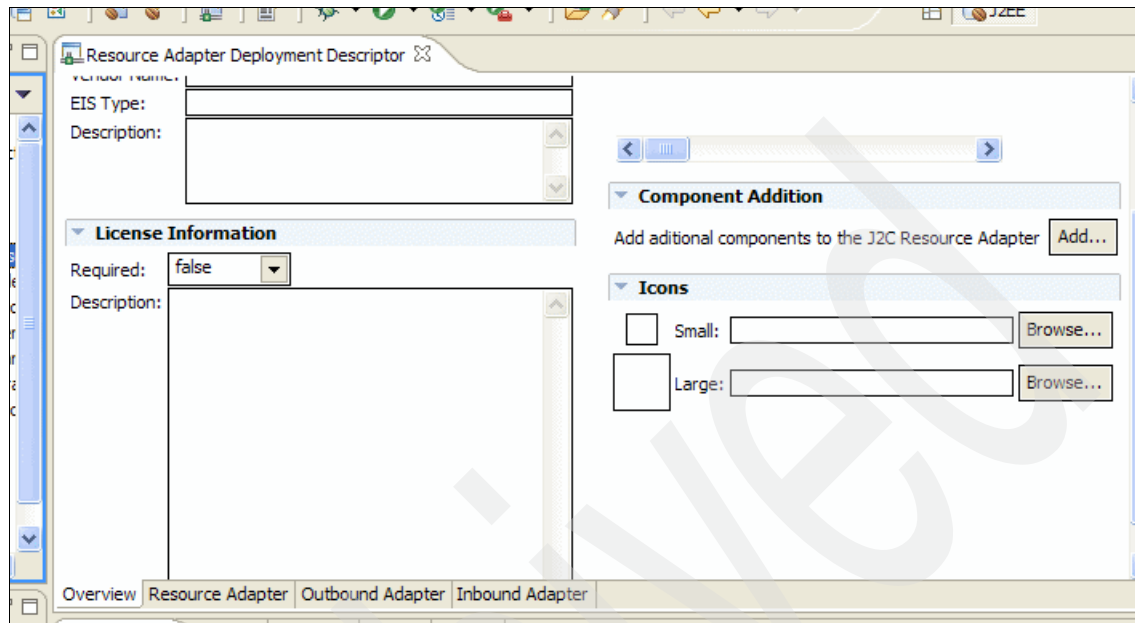


Figure 12-1 Adding inbound processing to an existent J2C adapter project

3. In the Add Component dialog box, select **Generate Inbound Adapter classes** and click **Finish**, as shown in Figure 12-2.

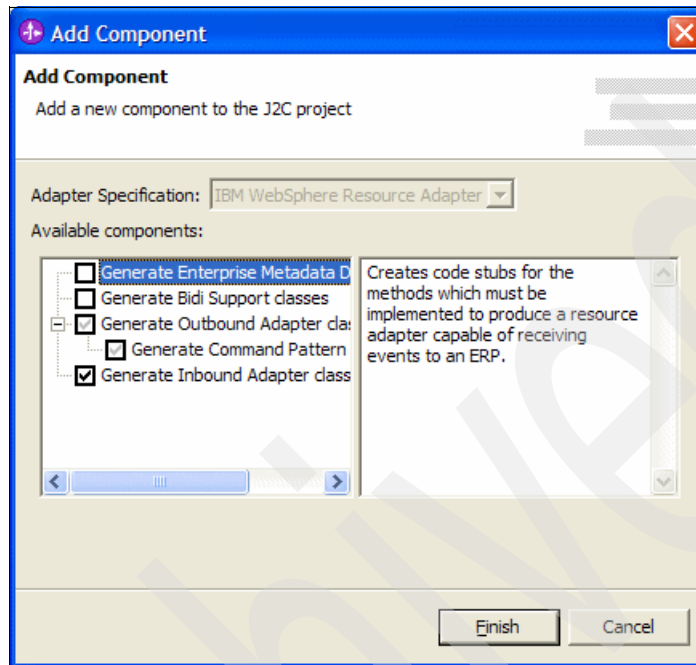


Figure 12-2 Selecting the Inbound adapter classes

4. The generated inbound classes now can be viewed in the J2EE perspective, under `com.ibm.itso.sab511.inbound` package.

12.2 Identify inbound events and properties

This section discusses how to identify inbound events and properties.

12.2.1 Inbound events

We need to identify EIS events that are of interest to the event consumer before implementing inbound processing for the adapter. Typically, the event consumers are application components running in the application server. Refer to Chapter 10, "Adapter design and specification" on page 237 for detail.

We are only interested in events that are related to Maintenance entity in RedMaintenance application. This information is important for adapter developers because it determines the development efforts that are involved.

RedMaintenance has a built-in event table. This table records the events that occurred on the Maintenance entity. RedMaintenance also offers an API to access this event store. This API allows the adapter to perform Create, Retrieve, Update, Delete (CRUD) operations on the event table during inbound event management.

When creating an adapter for your EIS, you typically create a event table and add database triggers. The triggers need to insert a event record to the event table for every events that you want to send to the adapter for inbound processing.

12.2.2 Inbound properties

RedMaintenance application is a Remote Method Invocation (RMI) server. To enable RedMaintenance adapter to connect to the RedMaintenance server, we need to identify the RMI URL. This to allow RedMaintenance adapter to bind to RedMaintenance server objects. We use the variable *rmiUrl* to represent this value as seen in Table 12-1.

Table 12-1 Inbound custom properties

RedMaintenance Inbound Properties	Values
rmiUrl	//localhost/rm

In Chapter 5, “Inbound processing and events” on page 107, we mentioned there are a set of standard Inbound properties. The RedMaintenance adapter uses the built-in default values for these standard properties. Refer to Chapter 5, “Inbound processing and events” on page 107 for more detail.

Here we show you how to create this property in the RedMaintenance adapter project:

1. In the J2EE Perspective, open the **Adapter Deployment Descriptor** using the Deployment Descriptor Editor.
2. In the Resource Adapter Deployment Descriptor, select the **Inbound Adapter pane** and then select the

commonj.connector.runtime.InboundListener message listener and in the properties list, click **Add** as Figure 12-3 shows.

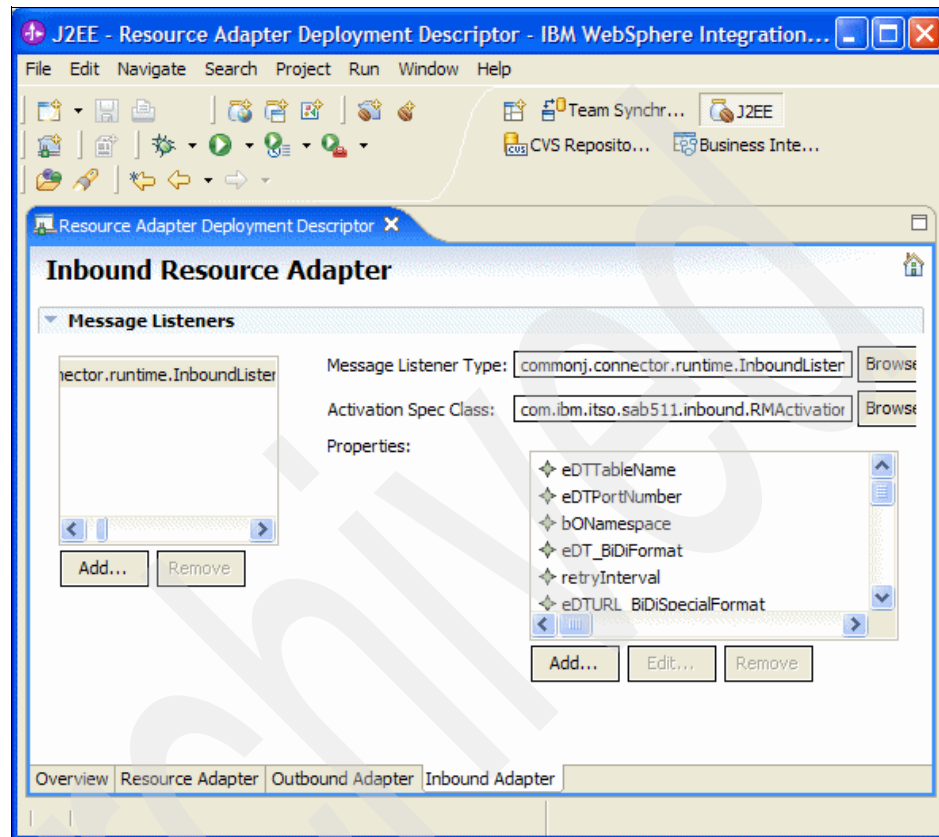


Figure 12-3 Inserting the custom inbound properties

3. In the Add Required Config Property, enter the following information in the Name, Type, Value, and Description fields:
 - rmiUrl
 - java.lang.String
 - //localhost/rm
 - RMI URL of RedMaintenance Application
4. Save the **Resource Adapter Deployment Descriptor** in order to let the WebSphere Integration Developer create the variables in the deployment descriptor file and the ActivationSpec subclass.

Important: WebSphere Adapter Toolkit automatically generates properties for the inbound event distribution table (eDTxxx properties). They are located in the activation spec section of adapter deployment descriptor. These properties must be set if you are not using the default event distribution table provided by Adapter Foundation classes. RedMaintenance adapter uses the default event distribution table. We do not need to set any values for these properties. However, if no values are set for these properties, RedMaintenance adapter is not able to deploy or start on WebSphere Process Server because the adapter deployment descriptor schema defines the activation spec properties as required properties. To overcome this, we must remove these properties manually from the adapter deployment descriptor using a text editor. When editing is complete, do not use the adapter deployment descriptor editor to open the adapter deployment descriptor. Otherwise, these deleted properties are regenerated. Example 12-1 shows an adapter deployment descriptor with event distribution table properties removed. Replace the content of the RedMaintenance adapter deployment descriptor with xml code in Example 12-1. This step is necessary if you are using WebSphere Adapter Toolkit 6.0 or 6.0.0.1 with WebSphere Process Server 6.0.1. This step might not be needed with future versions of WebSphere Adapter Toolkit.

Example 12-1 Adapter deployment descriptor ra.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<connector id="Connector_ID" version="1.5"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/connector_1_5.xsd">
  <display-name>
    RedMaintenance</display-name>
  <vendor-name></vendor-name>
  <eis-type></eis-type>
  <resourceadapter-version>1.0.0</resourceadapter-version>
  <license>
    <description>
    </description>
    <license-required>>false</license-required>
  </license>
  <resourceadapter>

  <resourceadapter-class>com.ibm.itso.sab511.RMResourceAdapter</resourceadapter-c
lass>
    <config-property>
      <config-property-name>adapterID</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
```



```

</config-property>
<config-property>
  <config-property-name>logFilename</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
</config-property>
<config-property>
  <config-property-name>biDiContextEIS</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
</config-property>
<config-property>
  <config-property-name>logNumberOfFiles</config-property-name>
  <config-property-type>java.lang.Integer</config-property-type>
</config-property>
<config-property>
  <config-property-name>threadContextPropagationRequired</config-property-name>
  <config-property-type>java.lang.Boolean</config-property-type>
</config-property>
<config-property>
  <config-property-name>biDiContextSpecialFormat</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
</config-property>
<config-property>
  <config-property-name>traceFileSize</config-property-name>
  <config-property-type>java.lang.Integer</config-property-type>
</config-property>
<config-property>
  <config-property-name>traceFilename</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
</config-property>
<config-property>
  <config-property-name>biDiContextSkip</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
</config-property>
<config-property>
  <config-property-name>logFileSize</config-property-name>
  <config-property-type>java.lang.Integer</config-property-type>
</config-property>
<config-property>
  <config-property-name>biDiContextMetadata</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
</config-property>
<config-property>
  <config-property-name>biDiContextTurnBiDiOff</config-property-name>
  <config-property-type>java.lang.Boolean</config-property-type>
</config-property>
<config-property>
  <config-property-name>traceNumberOfFiles</config-property-name>
  <config-property-type>java.lang.Integer</config-property-type>

```

```

    </config-property>
    <outbound-resourceadapter>
      <connection-definition>

        <managedconnectionfactory-class>com.ibm.itso.sab511.outbound.RMManagedConnectio
nFactory</managedconnectionfactory-class>
        <config-property>
          <config-property-name>password</config-property-name>
          <config-property-type>java.lang.String</config-property-type>
        </config-property>
        <config-property>
          <config-property-name>userName</config-property-name>
          <config-property-type>java.lang.String</config-property-type>
        </config-property>
        <config-property>

          <config-property-name>biDiContext_PasswordSkip</config-property-name>
          <config-property-type>java.lang.String</config-property-type>
        </config-property>
        <config-property>
          <config-property-name>biDiContextEIS</config-property-name>
          <config-property-type>java.lang.String</config-property-type>
        </config-property>
        <config-property>

          <config-property-name>biDiContext_UserNameSkip</config-property-name>
          <config-property-type>java.lang.String</config-property-type>
        </config-property>
        <config-property>

          <config-property-name>biDiContextSpecialFormat</config-property-name>
          <config-property-type>java.lang.String</config-property-type>
        </config-property>
        <config-property>

          <config-property-name>biDiContext_UserNameEIS</config-property-name>
          <config-property-type>java.lang.String</config-property-type>
        </config-property>
        <config-property>
          <config-property-name>reauthSupported</config-property-name>
          <config-property-type>java.lang.Boolean</config-property-type>
          <config-property-value>false</config-property-value>
        </config-property>
        <config-property>
          <config-property-name>biDiContextSkip</config-property-name>
          <config-property-type>java.lang.String</config-property-type>
        </config-property>
        <config-property>

```

```

<config-property-name>biDiContextMetadata</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
</config-property>
<config-property>

<config-property-name>biDiContext_PasswordEIS</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
</config-property>
<config-property>
  <description>
  </description>
  <config-property-name>serverURL</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
  <config-property-value>localhost</config-property-value>
</config-property>
<config-property>
  <description>
  </description>
  <config-property-name>rmiName</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
  <config-property-value>rm</config-property-value>
</config-property>

<connectionfactory-interface>javax.resource.cci.ConnectionFactory</connectionfactory-interface>

<connectionfactory-impl-class>com.ibm.itso.sab511.outbound.RMConnectionFactory</connectionfactory-impl-class>

<connection-interface>javax.resource.cci.Connection</connection-interface>

<connection-impl-class>com.ibm.itso.sab511.outbound.RMConnection</connection-impl-class>
  </connection-definition>
  <transaction-support>NoTransaction</transaction-support>
  <reauthentication-support>>false</reauthentication-support>
</outbound-resourceadapter>
<inbound-resourceadapter>
  <messageadapter>
  <messagelistener>

<messagelistener-type>commonj.connector.runtime.InboundListener</messagelistener-type>

  <activation-spec>

<activation-spec-class>com.ibm.itso.sab511.inbound.RMActivationSpec</activation-spec-class>

  <required-config-property>

```

```

        <description>
        </description>
        <config-property-name>b0Namespace</config-property-name>
    </required-config-property>
    <required-config-property>
        <description>
        </description>

    <config-property-name>retryInterval</config-property-name>
    </required-config-property>
    <required-config-property>
        <description>
        </description>

    <config-property-name>pollQuantity</config-property-name>
    </required-config-property>
    <required-config-property>
        <description>
        </description>
        <config-property-name>pollPeriod</config-property-name>
    </required-config-property>
    <required-config-property>
        <description>
        </description>

    <config-property-name>deliveryType</config-property-name>
    </required-config-property>
    <required-config-property>
        <description>
        </description>
        <config-property-name>rmiUrl</config-property-name>
    </required-config-property>
    </activation-spec>
</message-listener>
</message-adapter>
</inbound-resource-adapter>
<security-permission>
    <security-permission-spec>grant {permission
java.util.logging.LoggingPermission "control";};</security-permission-spec>
    </security-permission>
<security-permission>
    <description></description>
    <security-permission-spec>grant { permission
java.security.AllPermission
"*";"";"";};</security-permission-spec>
    </security-permission>
</resource-adapter>
</connector>

```

12.3 Add dependent jar files

The resource adapter has dependencies on the following jars files:

- ▶ RedMaintenance's common.jar.
- ▶ Adapter foundation's CWYBS_AdapterFoundation.jar.
- ▶ Internationalization libraries icu4j_3_2.jar.

If you are continuing from the previous chapter, you already have these dependent jar added. Otherwise, you can follow the instruction in 11.3, "Generate outbound stub classes" on page 255 to add dependent files to the adapter project.

12.4 Connection to EIS for inbound processing

To process inbound events, the adapter must provide a way to connect to the EIS. In Example 12-2, we create the utility class `InboundConnectionFactory` for this purpose. This class offers the basic functionality to connect and disconnect from the RedMaintenance application. The package of this class is `com.ibm.itso.sab511.utils`.

Example 12-2 InboundConnectionFactory implementation

```
/*
 * Created on Nov 29, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.ibm.itso.sab511.utils;

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;

import com.ibm.itso.rm.common.RMServerRMIInterface;

/**
 * @author zabala
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class InboundConnectionFactory {
    public RMServerRMIInterface getConnection() throws RemoteException {
```

```

        String host = "localhost";
        String appLocation = "rm";
        String name = "/" + host + "/" + appLocation;
        return getConnection(name);
    }

    public RMTServerRMIIInterface getConnection(String url)
        throws RemoteException {
        if (System.getSecurityManager() == null)
            System.setSecurityManager(new RMISecurityManager());

        RMTServerRMIIInterface eisInterface = null;
        try {
            eisInterface = (RMTServerRMIIInterface) Naming.lookup(url);
            // Execute some method of RMTServerRMIIInterface
            System.out.println("Connected");
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (RemoteException e) {
            e.printStackTrace();
        } catch (NotBoundException e) {
            e.printStackTrace();
        }
        return eisInterface;
    }

    public void releaseConnection(RMTServerRMIIInterface connection) {
        connection = null;
    }
}

```

12.5 Implement the RMTEvent subclass

The RedMaintenance application uses an event entity object conformed by the following attributes:

- ▶ eventId
- ▶ objectName
- ▶ status
- ▶ verb
- ▶ key
- ▶ priority
- ▶ description

This implies that the base Event class is not enough to capture all the information that RedMaintenance application uses in his event management process.

An Event subclass called RMEvent and located in the com.ibm.itso.sab511 package is created for this reason.

Example 12-3 lists the implementation of RMEvent subclass, incorporating the verb attribute inside the new class.

Example 12-3 RMEvent implementation

```
/*
 * Created on Nov 29, 2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.ibm.itso.sab511.inbound;

import java.sql.Timestamp;

import com.ibm.j2ca.extension.eventmanagement.Event;

/**
 * @author zabala
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class RMEvent extends Event {
    private String verb;

    /**
     *
     */
    public RMEvent() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @param arg0
     * @param arg1
     * @param arg2
     */
    public RMEvent(String id, String type, int status) {
        super(id, type, status);
        // TODO Auto-generated constructor stub
    }
}
```

```

/**
 * @param arg0
 * @param arg1
 * @param arg2
 * @param arg3
 */
public RMEvent(String id, String type, String keys, int status) {
    super(id, type, keys, status);
    // TODO Auto-generated constructor stub
}

/**
 * @param arg0
 * @param arg1
 * @param arg2
 * @param arg3
 * @param arg4
 */
public RMEvent(String id, String type, String keys, int status,
    Timestamp timeStamp) {
    super(id, type, keys, status, timeStamp);
    // TODO Auto-generated constructor stub
}

/**
 * @return Returns the verb.
 */
public String getVerb() {
    return verb;
}

/**
 * @param verb The verb to set.
 */
public void setVerb(String verb) {
    this.verb = verb;
}
}

```

12.6 Implementation of RMEventStore class

Inbound processing can be quite complex. Fortunately, Adapter Foundation Classes have provided generic business logic to handle inbound processing. To leverage these inbound processing business logic, we must provide RedMaintenance specific implementations to overwrite related methods in the

super class. These methods are called by Adapter Foundation Classes during inbound processing.

12.6.1 Event filtering

The RedMaintenance event management does not have the ability to filter the events for a given search criteria, so this option must be turned off. Example 12-4 shows how this is done.

Example 12-4 RMEventStore.implementsFiltering

```
public boolean implementsFiltering() {  
    return false;  
}
```

12.6.2 Polling events

Example 12-5 shows RedMaintenance adapter calling RedMaintenance application's API to poll events from the event table. The business logic for polling is implemented in this method. This method is called by Adapter Foundation Classes inbound framework at runtime.

Example 12-5 RMEventStore.getEvents

```
/**  
 * Polls RedMaintenance's event store for events.  
 */  
public java.util.ArrayList getEvents(int pollQuantity, int status,  
    String[] eventTypeFilter) throws javax.resource.ResourceException,  
    javax.resource.spi.CommException {  
  
    ArrayList currentEvents = new ArrayList();  
    int currentFetchQuantity = 0;  
    try {  
        // 1. Get the connection to the EIS  
        traceFiner("getEvents()", "pollQuantity: " + pollQuantity  
            + ", status: " + status);  
        RMServerRMIInterface eisCurrentConnection = new  
InboundConnectionFactory()  
            .getConnection(activationSpec.getRmiUrl());  
        String eventId = "0";  
        boolean otherEvents = true;  
        int eventRetrieved = 0;  
        // 2. Search for events  
        for (; otherEvents && eventRetrieved < pollQuantity;) {  
            // A RMDDataImplementation instance must be obtained in order to  
            // begin the events' search
```

```

        RMDDataImplementation inDataImpl = getRMDDataImplementationInstance(
            eventId, status);
        try {
            RMDDataImplementation eisEvent = (RMDDataImplementation)
eisCurrentConnection
                .retrieveObject(inDataImpl);
            traceFine("getEvents()", "Event Found "
                + eisEvent.getAttr("Id") + ":"
                + eisEvent.getAttr("Status") + ":"
                + eisEvent.getAttr("Description"));

            // The eisEvent object must be traslated to an Adapter Event
            // instance
            RMEvent event = toAdapterEvent(eisEvent);
            eventId = event.getEventId();
            currentEvents.add(event);
            eventRetrieved++;
        } catch (RemoteException re) {
            traceFinest("getEvents()",
                "No more events to process in EIS event table");
            otherEvents = false;
        }
    }
    currentFetchQuantity = Math.min(eventRetrieved, pollQuantity);
    logger.trace(Level.FINE, RMEventStore.class.getName(),
        "getEvents()", "Found " + eventRetrieved
            + " events to process");
    } catch (RemoteException re2) {
        logConnectFailed(activationSpec.getRmiUrl());
        throw new ResourceException(re2);
    }
    ArrayList sortedList = sortEvents(currentEvents);
    List subList = sortedList.subList(0, currentFetchQuantity);
    ArrayList returnableList = new ArrayList(currentFetchQuantity);
    returnableList.addAll(subList);
    traceFiner("getEvents()", "End processing: returnableList: "
        + returnableList.size());
    return returnableList;
}

```

12.6.3 Deleting events

Example 12-6 shows the implementation of `deleteEvent`. This method is called by Adapter Foundation Classes Framework at after an event has been successfully delivered.

```
/**
 * Delete an event from RedMaintenance's event table
 */
public void deleteEvent(com.ibm.j2ca.extension.eventmanagement.Event evt)
    throws javax.resource.ResourceException,
           javax.resource.spi.CommException {

    traceMethodEntrance("deleteEvent()");
    try {
        traceFiner("deleteEvent()", "deleting event with ID"
            + evt.getEventId());
        RMDDataImplementation eisEvent = toEISEvent(evt);
        connection.deleteObject(eisEvent);
    } catch (RemoteException e) {
        logDeleteFailed(evt.getEventId());
        logCommFailure("deleteEvent()");
        throw new CommException(e);
    }
    traceMethodExit("deleteEvent()");
}
```

12.6.4 Updating events

Example 12-7 shows the implementation of `updateEvent`. This method is called by Adapter Foundation Classes Framework to update the status of a event.

```
/**
 * Update the event status in RedMaintenance's event table
 */
public void updateEventStatus(
    com.ibm.j2ca.extension.eventmanagement.Event event, int newStatus)
    throws javax.resource.ResourceException,
           javax.resource.spi.CommException {

    traceMethodEntrance("updateEventStatus()");
    traceFiner("updateEventStatus()", "updating event:"
        + event.getEventId() + " to status:" + newStatus);
    try {
        RMDDataImplementation eisEvent = toEISEvent(event);
        eisEvent.setAttr("Status", String.valueOf(newStatus));
        connection.updateObject(eisEvent);
        traceFiner("updateEventStatus()", "updating event:"
            + event.getEventId() + " to status:" + newStatus
            + ": Successful.");
    }
```

```

    } catch (RemoteException e) {
        logUpdateFailed(event.getEventId());
        logCommFailure("updateEventStatus()");
        throw new CommException(e);
    }
    traceMethodExit("updateEventStatus()");
}

```

12.6.5 Retrieving events

Example 12-8 shows the implementation of event retrieval. This method is called by Adapter Foundation Classes Framework to retrieve a specific event.

Example 12-8 RMEventStore.getSpecificEvent

```

/**
 * Get an specific event from RedMaintenance's event table
 */
public com.ibm.j2ca.extension.eventmanagement.Event getSpecificEvent(
    java.lang.String eventId) throws javax.resource.ResourceException,
    javax.resource.spi.CommException {
    traceMethodEntrance("getSpecificEvent()");
    traceFiner("getSpecificEvent()", "getting event with ID: " + eventId);
    RMEvent currentEvent;
    RMDDataImplementation inDataImpl =
getRMDDataImplementationInstance(eventId);
    try {
        RMDDataImplementation eisEvent = (RMDDataImplementation) connection
            .retrieveObject(inDataImpl);
        traceFinest("getSpecificEvent()", "Event Found "
            + eisEvent.getAttr("Id") + ":" + eisEvent.getAttr("Status")
            + ":" + eisEvent.getAttr("Description"));
        // The eisEvent object must be traslated to an Adapter Event
        // instance
        currentEvent = toAdapterEvent(eisEvent);
    } catch (RemoteException e) {
        logGetSpecificEventFailed(eventId);
        logCommFailure("getSpecificEvent()");
        throw new CommException(e);
    } finally {
        traceMethodExit("getSpecificEvent()");
    }
    return currentEvent;
}

```

12.6.6 Retrieving business objects

When implementing `getObjectForEvent` method, we can call outbound processing's retrieve command. This command is used to get the business object related to the event specified in the event store. The retrieve command generates a corresponding hierarchical business object if the event object is hierarchical.

The code listed in Example 12-9 shows the implementation of the `getObjectForEvent` method.

Example 12-9 RMEventStore.getObjectForEvent

```
/**
 * From an event object, retrieve the RedMaintenance's data object from
 * RedMaintenance and return it's business graph
 */
public synchronized Object getObjectForEvent(Event event)
    throws ResourceException {

    RMEvent rmEvent = (RMEvent) event;
    traceMethodEntrance("getObjectForEvent()");
    traceFiner("getObjectForEvent()",
        "attempting to retrieve full object for event:" + event);
    String type = event.getEventType();
    String bgType = type + "BG";
    traceFiner("getObjectForEvent()", "object type:" + bgType);

    DataObject bgObject;
    try {
        //create a business graph
        bgObject = AdapterBOUtil.createDataObject(boNamespace + "/"
            + bgType.toLowerCase(), bgType);
        traceFiner("getObjectForEvent()", "NEW: bgObject created! : "
            + bgObject.getType().getName());
    } catch (BusinessObjectDefinitionNotFoundException e) {
        logBONotFound(e);
        throw e;
    }
    Property property = AdapterBOUtil
        .getRootBusinessObjectProperty(bgObject.getType());
    traceFinest("getObjectForEvent()", "Property: name:"
        + property.getName() + ", type:" + property.getType().getName());

    //create a business object
    DataObject dataObject = bgObject.createDataObject(property);
    traceFinest("getObjectForEvent()", "dataObject created: "
        + dataObject.getType().getName());
}
```

```

objectNaming.setKey(dataObject, event.getEventKeys());
traceFinest("getObjectForEvent()", "setKey of dataObject success!");

//get a retrieve command
RMRetrieveCommand retrieveCommand = (RMRetrieveCommand) commandFactory
    .createCommand(WBIInteractionSpec.RETRIEVE_OP, dataObject);
traceFinest("getObjectForEvent()", "Retrieve command created");
retrieveCommand.setConnection(connection);
retrieveCommand.setLogUtils(logger);

//execute retrieve command to retrieve the data from RedMaintenance
DataObject returnedObject = retrieveCommand.execute(dataObject);
traceFinest("getObjectForEvent()", "Retrieve command executed");

//add the business object to the business graph container
bgObject.setDataObject(AdapterBOUtil
    .getRootBusinessObjectProperty(bgObject.getType()),
    returnedObject);
traceFinest("getObjectForEvent()",
    "returnedObject inserted into bgObject");

//set the business graph's verb
bgObject.set("verb", rmEvent.getVerb());
WBIRecord outputRecord = new WBIRecord();

//JCA spec specifies that we must return a Record object
//so we use a Record object to wrap the business graph
outputRecord.setDataObject(bgObject);
traceFinest("getObjectForEvent()", "ouputRecord created!");
traceMethodExit("getObjectForEvent()");
return outputRecord;
}

```

12.6.7 Transactional Considerations

RedMaintenance does not have a transactional management of events, so all the methods regarding this topic have to be coded accordingly (returning false as the value of the isTransactional method and leaving blank the implementation of commitWork and rollbackWork methods). Example 12-10 shows their implementation.

Example 12-10 RMEventStore transactional methods

```

public boolean isTransactional() {
    // TODO Auto-generated method stub
    return false;
}

```

```

    public void rollbackWork() throws javax.resource.ResourceException,
        javax.resource.spi.CommException {
        // TODO Auto-generated method stub
    }

    public void commitWork() throws javax.resource.ResourceException,
        javax.resource.spi.CommException {
        // TODO Auto-generated method stub
    }

```

12.6.8 Full code of RMEventStore

Example 12-11 lists the complete implementation of *RMEventStore*.

Example 12-11 RMEventStore implementation

```

package com.ibm.itso.sab511.inbound;

import java.rmi.RemoteException;
import java.sql.Timestamp;
import java.text.ParseException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.logging.Level;

import javax.resource.ResourceException;
import javax.resource.spi.CommException;

import com.ibm.itso.rm.common.RMDataImplementation;
import com.ibm.itso.rm.common.RMServerRMIInterface;
import com.ibm.itso.sab511.RMResourceAdapter;
import com.ibm.itso.sab511.outbound.commandpattern.RMCommandFactoryImpl;
import com.ibm.itso.sab511.outbound.commandpattern.RMRetrieveCommand;
import com.ibm.itso.sab511.utils.InboundConnectionFactory;
import com.ibm.itso.sab511.utils.ObjectNaming;
import com.ibm.j2ca.base.AdapterBOUtil;
import com.ibm.j2ca.base.WBIInteractionSpec;
import com.ibm.j2ca.base.WBIRecord;
import com.ibm.j2ca.base.exceptions.BusinessObjectDefinitionNotFoundException;
import com.ibm.j2ca.extension.eventmanagement.Event;
import com.ibm.j2ca.extension.eventmanagement.EventStore;
import com.ibm.j2ca.extension.logging.LogUtilConstants;
import com.ibm.j2ca.extension.logging.LogUtils;
import commonj.sdo.DataObject;
import commonj.sdo.Property;

```

```

/**
 * EMEventStore class provides methods to poll RedMaintenance's event store.
 *
 */
public class RMEventStore implements EventStore {
    String boNamespace;

    RMActivationSpec activationSpec;

    RMResourceAdapter resourceAdapter;

    RMServerRMIInterface connection;

    LogUtils logger;

    ObjectNaming objectNaming;

    RMCommandFactoryImpl commandFactory;

    /**
     * Initialize class variables
     *
     * @param activationSpec
     * @param resourceAdapter
     * @throws ResourceException
     */
    public RMEventStore(RMActivationSpec activationSpec,
        RMResourceAdapter resourceAdapter) throws ResourceException {
        super();

        this.activationSpec = activationSpec;
        logger = resourceAdapter.getLogUtils();
        boNamespace = activationSpec.getBONamespace();
        try {
            connection = new InboundConnectionFactory()
                .getConnection(activationSpec.getRmiUrl());
        } catch (RemoteException e) {
            e.printStackTrace();
            throw new CommException(e);
        }
        this.resourceAdapter = resourceAdapter;
        objectNaming = new ObjectNaming(resourceAdapter);
        commandFactory = new RMCommandFactoryImpl(objectNaming);
    }

    /**
     * (non-Javadoc)
     *

```



```

    * @see
com.ibm.j2ca.extension.eventmanagement.EventStore#implementsFiltering()
    */
    public boolean implementsFiltering() {
        // TODO Auto-generated method stub
        return false;
    }

    /**
     * Polls RedMaintenance's event store for events.
     */
    public java.util.ArrayList getEvents(int pollQuantity, int status,
        String[] eventTypeFilter) throws javax.resource.ResourceException,
        javax.resource.spi.CommException {

        ArrayList currentEvents = new ArrayList();
        int currentFetchQuantity = 0;
        try {
            // 1. Get the connection to the EIS
            traceFiner("getEvents()", "pollQuantity: " + pollQuantity
                + ", status: " + status);
            RMServerRMIInterface eisCurrentConnection = new
                InboundConnectionFactory()
                    .getConnection(activationSpec.getRmiUrl());
            String eventId = "0";
            boolean otherEvents = true;
            int eventRetrieved = 0;
            // 2. Search for events
            for (; otherEvents && eventRetrieved < pollQuantity; ) {
                // A RMDDataImplementation instance must be obtained in order to
                // begin the events' search
                RMDDataImplementation inDataImpl = getRMDDataImplementationInstance(
                    eventId, status);
                try {
                    RMDDataImplementation eisEvent = (RMDDataImplementation)
eisCurrentConnection
                        .retrieveObject(inDataImpl);
                    traceFine("getEvents()", "Event Found "
                        + eisEvent.getAttr("Id") + ":"
                        + eisEvent.getAttr("Status") + ":"
                        + eisEvent.getAttr("Description"));

                    // The eisEvent object must be traslated to an Adapter Event
                    // instance
                    RMEvent event = toAdapterEvent(eisEvent);
                    eventId = event.getEventId();
                    currentEvents.add(event);
                    eventRetrieved++;
                } catch (RemoteException re) {

```

```

        traceFinest("getEvents()",
            "No more events to process in EIS event table");
        otherEvents = false;
    }
}
currentFetchQuantity = Math.min(eventRetrieved, pollQuantity);
logger.trace(Level.FINE, RMEventStore.class.getName(),
    "getEvents()", "Found " + eventRetrieved
        + " events to process");
} catch (RemoteException re2) {
    logConnectFailed(activationSpec.getRmiUrl());
    throw new ResourceException(re2);
}
ArrayList sortedList = sortEvents(currentEvents);
List subList = sortedList.subList(0, currentFetchQuantity);
ArrayList returnableList = new ArrayList(currentFetchQuantity);
returnableList.addAll(subList);
traceFiner("getEvents()", "End processing: returnableList: "
    + returnableList.size());
return returnableList;
}

/**
 * Sort events base on the event's creation time stamp
 *
 * @param currentEvents
 * @return
 */
private ArrayList sortEvents(ArrayList currentEvents) {
    ArrayList sortedEvents = (ArrayList) currentEvents.clone();
    Collections.sort(sortedEvents, new Comparator() {
        public int compare(Object event1, Object event2) {
            return isAfter((Event) event1, (Event) event2);
        }
    });
    return sortedEvents;
}

private int isAfter(Event event1, Event event2) {
    if (event1.getTimeStamp().after(event2.getTimeStamp()))
        return 1;
    return 0;
}

/**
 * Convert a RedMaintenance native event object to a RMEvent object.
 *
 * @param eisEvent
 * @return

```

```

*/
private RMEvent toAdapterEvent(RMDataImplementation eisEvent) {
    // A new RMEvent must be generated from the eisEvent
    RMEvent event = new RMEvent();

    event.setEventId(eisEvent.getAttr("Id"));
    event.setEventKeys(eisEvent.getAttr("Key"));
    // RedMaintenance App does put an additional "RM_" Prefix
    // to the ObjectName attribute.. so, we have to delete it
    String objectName = eisEvent.getAttr("ObjectName").startsWith("RM_") ?
eisEvent
        .getAttr("ObjectName").substring(3)
        : eisEvent.getAttr("ObjectName");
    event.setEventType(objectName);
    long timestamp;
    try {
        timestamp = eisEvent.getLongAttr("EventTimeStamp");
    } catch (ParseException e) {
        timestamp = System.currentTimeMillis();
    }
    event.setEventStatus(Event.NEWEVENT);
    event.setTimeStamp(new Timestamp(timestamp));
    event.setVerb(eisEvent.getAttr("Verb"));
    traceFinest("getAdapterEventFromEisEvent()", "RMEvent created: Id: "
        + event.getEventId() + ", keys: " + event.getEventKeys()
        + ", eventType: " + event.getEventType() + ", status: "
        + event.getEventStatus() + ", verb: " + event.getVerb());
    return event;
}

/**
 * Get an instance of RedMaintenance data object for given event id and
 * event status
 *
 * @param eventId
 * @param status
 * @return
 */
private RMDataImplementation getRMDataImplementationInstance(
    String eventId, int status) {
    RMDataImplementation data = getRMDataImplementationInstance(eventId);
    data.setAttr("Status", String.valueOf(status));
    return data;
}

/**
 * Get a new instance of RedMaintenance data object for an event id
 *
 * @param eventId

```

```

        * @return
        */
        private RMDDataImplementation getRMDDataImplementationInstance(String
eventId) {
            traceMethodEntrance("getRMDDataImplementationInstance()");
            RMDDataImplementation aDataImpl = new RMDDataImplementation();
            aDataImpl.setComponentName("Event");
            aDataImpl.setAttr("Id", eventId);
            traceMethodExit("getRMDDataImplementationInstance()");
            return aDataImpl;
        }

/**
 * Delete an event from RedMaintenance's event table
 */
public void deleteEvent(com.ibm.j2ca.extension.eventmanagement.Event evt)
    throws javax.resource.ResourceException,
        javax.resource.spi.CommException {

    traceMethodEntrance("deleteEvent()");
    try {
        traceFiner("deleteEvent()", "deleting event with ID"
            + evt.getEventId());
        RMDDataImplementation eisEvent = toEISEvent(evt);
        connection.deleteObject(eisEvent);
    } catch (RemoteException e) {
        logDeleteFailed(evt.getEventId());
        logCommFailure("deleteEvent()");
        throw new CommException(e);
    }
    traceMethodExit("deleteEvent()");
}

/**
 * Convert a RMevent to a native RedMaintenance event.
 *
 * @param evt
 * @return
 */
private RMDDataImplementation toEISEvent(Event adapterEvent) {
    RMDDataImplementation eisEvent = new RMDDataImplementation();
    eisEvent.setComponentName("Event");
    eisEvent.setAttr("Id", adapterEvent.getEventId());
    eisEvent.setAttr("Key", adapterEvent.getEventKeys());
    eisEvent.setAttr("Status", adapterEvent.getEventStatus());
    //RedMaintenance App needs the "RM_" Prefix
    eisEvent.setAttr("ObjectName", "RM_" + adapterEvent.getEventType());
    return eisEvent;
}

```

```

/**
 * Update the event status in RedMaintenance's event table
 */
public void updateEventStatus(
    com.ibm.j2ca.extension.eventmanagement.Event event, int newStatus)
    throws javax.resource.ResourceException,
    javax.resource.spi.CommException {

    traceMethodEntrance("updateEventStatus()");
    traceFiner("updateEventStatus()", "updating event:"
        + event.getEventId() + " to status:" + newStatus);
    try {
        RMDDataImplementation eisEvent = toEISEvent(event);
        eisEvent.setAttr("Status", String.valueOf(newStatus));
        connection.updateObject(eisEvent);
        traceFiner("updateEventStatus()", "updating event:"
            + event.getEventId() + " to status:" + newStatus
            + ": Successfull.");
    } catch (RemoteException e) {
        logUpdateFailed(event.getEventId());
        logCommFailure("updateEventStatus()");
        throw new CommException(e);
    }
    traceMethodExit("updateEventStatus()");
}

/**
 * Get an specific event from RedMaintenance's event table
 */
public com.ibm.j2ca.extension.eventmanagement.Event getSpecificEvent(
    java.lang.String eventId) throws javax.resource.ResourceException,
    javax.resource.spi.CommException {
    traceMethodEntrance("getSpecificEvent()");
    traceFiner("getSpecificEvent()", "getting event with ID: " + eventId);
    RMEvent currentEvent;
    RMDDataImplementation inDataImpl =
getRMDDataImplementationInstance(eventId);
    try {
        RMDDataImplementation eisEvent = (RMDDataImplementation) connection
            .retrieveObject(inDataImpl);
        traceFinest("getSpecificEvent()", "Event Found "
            + eisEvent.getAttr("Id") + ":" + eisEvent.getAttr("Status")
            + ":" + eisEvent.getAttr("Description"));
        // The eisEvent object must be traslated to an Adapter Event
        // instance
        currentEvent = toAdapterEvent(eisEvent);
    } catch (RemoteException e) {
        logGetSpecificEventFailed(eventId);
    }
}

```

```

        logCommFailure("getSpecificEvent()");
        throw new CommException(e);
    } finally {
        traceMethodExit("getSpecificEvent()");
    }
    return currentEvent;
}

/**
 * From an event object, retrieve the RedMaintenance's data object from
 * RedMaintenance and return it's business graph
 */
public synchronized Object getObjectForEvent(Event event)
    throws ResourceException {

    RMEvent rmEvent = (RMEvent) event;
    traceMethodEntrance("getObjectForEvent()");
    traceFiner("getObjectForEvent()",
        "attempting to retrieve full object for event:" + event);
    String type = event.getEventType();
    String bgType = type + "BG";
    traceFiner("getObjectForEvent()", "object type:" + bgType);

    DataObject bgObject;
    try {
        //create a business graph
        bgObject = AdapterBOUtil.createDataObject(boNamespace + "/"
            + bgType.toLowerCase(), bgType);
        traceFiner("getObjectForEvent()", "NEW: bgObject created! : "
            + bgObject.getType().getName());
    } catch (BusinessObjectDefinitionNotFoundException e) {
        logBONotFound(e);
        throw e;
    }
    Property property = AdapterBOUtil
        .getRootBusinessObjectProperty(bgObject.getType());
    traceFinest("getObjectForEvent()", "Property: name:"
        + property.getName() + ", type:" + property.getType().getName());

    //create a business object
    DataObject dataObject = bgObject.createDataObject(property);
    traceFinest("getObjectForEvent()", "dataObject created: "
        + dataObject.getType().getName());

    objectNaming.setKey(dataObject, event.getEventKeys());
    traceFinest("getObjectForEvent()", "setKey of dataObject success!");

    //get a retrieve command

```

```

RMRetrieveCommand retrieveCommand = (RMRetrieveCommand) commandFactory
    .createCommand(WBIInteractionSpec.RETRIEVE_OP, dataObject);
traceFinest("getObjectForEvent()", "Retrieve command created");
retrieveCommand.setConnection(connection);
retrieveCommand.setLogUtils(logger);

//execute retrieve command to retrieve the data from RedMaintenance
DataObject returnedObject = retrieveCommand.execute(dataObject);
traceFinest("getObjectForEvent()", "Retrieve command executed");

//add the business object to the business graph container
bgObject.setDataObject(AdapterBOUtil
    .getRootBusinessObjectProperty(bgObject.getType()),
    returnedObject);
traceFinest("getObjectForEvent()",
    "returnedObject inserted into bgObject");

//set the business graph's verb
bgObject.set("verb", rmEvent.getVerb());
WBIRRecord outputRecord = new WBIRRecord();

//JCA spec specifies that we must return a Record object
//so we use a Record object to wrap the business graph
outputRecord.setDataObject(bgObject);
traceFinest("getObjectForEvent()", "ouputRecord created!");
traceMethodExit("getObjectForEvent()");
return outputRecord;
}

/*
 * (non-Javadoc)
 *
 * @see com.ibm.j2ca.extension.eventmanagement.EventStore#isTransactional()
 */
public boolean isTransactional() {
    // TODO Auto-generated method stub
    return false;
}

/*
 * (non-Javadoc)
 *
 * @see com.ibm.j2ca.extension.eventmanagement.EventStore#rollbackWork()
 */
public void rollbackWork() throws javax.resource.ResourceException,
    javax.resource.spi.CommException {
    // TODO Auto-generated method stub
}

```

```

/*
 * (non-Javadoc)
 *
 * @see com.ibm.j2ca.extension.eventmanagement.EventStore#commitWork()
 */
public void commitWork() throws javax.resource.ResourceException,
    javax.resource.spi.CommException {
    // TODO Auto-generated method stub

}

private void traceMethodEntrance(String methodName) {
    logger.traceMethodEntrance(RMEventStore.class.getName(), methodName);
}

private void traceMethodExit(String methodName) {
    logger.traceMethodExit(RMEventStore.class.getName(), methodName);
}

private void traceFine(String methodName, String message) {
    logger.trace(Level.FINE, RMEventStore.class.getName(), methodName,
        message);
}

private void traceFiner(String methodName, String message) {
    logger.trace(Level.FINER, RMEventStore.class.getName(), methodName,
        message);
}

private void traceFinest(String methodName, String message) {
    logger.trace(Level.FINEST, RMEventStore.class.getName(), methodName,
        message);
}

private void logGetEventsFailed(Exception e) {
    logger.trace(Level.SEVERE, RMEventStore.class.getName(), "getEvents()",
        "getEvents() failed. Exception:" + e.getClass().getName()
        + ":" + e.getMessage());
    logger.log(Level.SEVERE, LogUtilConstants.ADAPTER_RBUNDLE,
        RMEventStore.class.getName(), "getEvents()", "1002",
        new Object[] {});
}

private void logDeleteFailed(String eventId) {
    logger.trace(Level.SEVERE, RMEventStore.class.getName(),
        "deleteEvent()", "delete failed for event id:" + eventId);
    logger.log(Level.SEVERE, LogUtilConstants.ADAPTER_RBUNDLE,
        RMEventStore.class.getName(), "deleteEvent()", "1003",

```



```

        new Object[] { eventID });
    }

    private void logUpdateFailed(String eventID) {
        logger.trace(Level.SEVERE, RMEventStore.class.getName(),
            "updateEvent()", "update failed for event id:" + eventID);
        logger.log(Level.SEVERE, LogUtilConstants.ADAPTER_RBUNDLE,
            RMEventStore.class.getName(), "updateEvent()", "1006",
            new Object[] { eventID });
    }

    private void logGetSpecificEventFailed(String eventID) {
        logger.trace(Level.SEVERE, RMEventStore.class.getName(),
            "getSpecificEvent()", "getSpecificEvent failed for event id:"
                + eventID);
        logger.log(Level.SEVERE, LogUtilConstants.ADAPTER_RBUNDLE,
            RMEventStore.class.getName(), "getSpecificEvent()", "1006",
            new Object[] { eventID });
    }

    private void logCommFailure(String methodName) {
        logger.log(Level.SEVERE, LogUtilConstants.ADAPTER_RBUNDLE,
            RMEventStore.class.getName(), methodName, "1004",
            new Object[] {});
    }

    private void logConnectFailed(String url) {
        logger.trace(Level.SEVERE, RMEventStore.class.getName(), "Constructor",
            "unable to connect to RM at URL:" + url);
        logger.log(Level.SEVERE, LogUtilConstants.ADAPTER_RBUNDLE,
            RMEventStore.class.getName(), "getEvents()", "1002",
            new Object[] {});
    }

    private void logBONotFound(BusinessObjectDefinitionNotFoundException e) {
        logger.trace(Level.SEVERE, RMEventStore.class.getName(),
            "getObjectForEvent()",
            "unable to create business object for type" + e.getNamespace()
                + "/" + e.getType());
        logger.log(Level.SEVERE, LogUtilConstants.ADAPTER_RBUNDLE,
            RMEventStore.class.getName(), "getObjectForEvent()", "1009",
            new Object[] { e.getNamespace() + "/" + e.getType() });
    }
}

```

12.7 Revision of the RMActivationSpec class

As mentioned earlier, the RMActivationSpec class must:

- ▶ Extend the `com.ibm.j2ca.base.WBIActivationSpec` class.
- ▶ Implement all the customized properties defined in the deployment descriptor of the resource adapter. For RedMaintenance, the property needed is `rmiUrl`.

Example 12-12 shows the RMActivationSpec implementation.

Example 12-12 RMActivationSpec implementation

```
package com.ibm.itso.sab511.inbound;

import com.ibm.j2ca.base.WBIActivationSpec;

public class RMActivationSpec extends WBIActivationSpec {
    private String rmiUrl = "//localhost/rm";
    /**
     *
     */
    public RMActivationSpec() {
        super();
        // TODO Auto-generated constructor stub
    }

    public void setRmiUrl(String newValue) {
        this.rmiUrl = newValue;
    }
    public String getRmiUrl() {
        return this.rmiUrl;
    }
    /** (non-Javadoc)
     * @see com.ibm.j2ca.base.WBIActivationSpec#getBONamespace()
     */
    public String getBONamespace() {
        // TODO Auto-generated method stub
        return super.getBONamespace();
    }
    /** (non-Javadoc)
     * @see
     com.ibm.j2ca.base.WBIActivationSpec#setBONamespace(java.lang.String)
     */
    public void setBONamespace(String arg0) {
        // TODO Auto-generated method stub
        super.setBONamespace(arg0);
    }
}
```

Tip: To avoid compilation problems in the code, you should overwrite the `getBONamespace` and `setBONamespace` methods in the implementation of the custom `ActivationSpec` subclass.

12.8 Change of `RMResourceAdapter` class

The `RMResourceAdapter` has to overwrite the `createEventStore` method in order to link the resource adapter with the implementation of the Event Store that have been developed. Example 12-13 shows the implementation of this method.

Example 12-13 `RMResourceAdapter.createEventStore`

```
public com.ibm.j2ca.extension.eventmanagement.EventStore createEventStore(
    javax.resource.spi.ActivationSpec activationSpec)
    throws javax.resource.ResourceException {
    // TODO Auto-generated method stub
    EventStore eventStore = new RMEventStore((RMActivationSpec)
activationSpec, this);
    return eventStore;
}
```

12.9 Creation of the SCA artifacts

In order to test the inbound processing of the RedMaintenance custom adapter, we need to provide the appropriated SCA artifacts to expose the adapter services to WebSphere Process Server. These artifacts can be automatically generated by Enterprise Service Discovery (ESD). Given that we have not discussed EMD implementation, we created these artifacts manually. The process we explain here is similar to 11.7, “Create temporary SCA artifacts” on page 297, but there are some minor differences:

1. Maintenance, and PartOrder business objects

These business objects contain application-specific information that our custom adapter must have to process the business objects in event management.

2. RedMaintenance ASI schema

This schema constrains the application-specific business information annotation tags in the business object definitions.

3. `discovery-service.xml` file

This file specifies, among other things, the namespace and location of RedMaintenance ASI schema.

4. RedMaintenanceInboundInterface.export file

This SCA EIS export binding. file binds the adapter inbound processing to SCA in WebSphere Process Server.

Use WebSphere Integration Developer to generate the following:

1. Maintenance, and PartOrder business graphs
2. RedMaintenanceInboundInterface.wsdl interface file

12.9.1 Provide the basic business objects and related files

The RedMaintenance adapter needs Maintenance and PartOrder business object definitions for inbound process testing. The RedMaintenance ASI schema specifies the metadata that is allowed in the business objects. Create these files as described below:

1. Create Maintenance.xsd and PartOrder.xsd files as described in the 11.7.1, “Create business objects with application-specific information” on page 298.
2. Create the RedMaintenance ASI schema as described in the 11.7.2, “Creating RedMaintenance ASI schema” on page 311.
3. Create the discovery-service.xml file as described in the 11.7.3, “Create discovery-service.xml file” on page 313.
4. Create the Maintenance Business Graph as described in 11.7, “Create temporary SCA artifacts” on page 297.

12.9.2 Create the RedMaintenance inbound interface

The inbound interface is the interface in which the RedMaintenance adapter exposes the operations that an application component has to implement in order to subscribe as a message endpoint. Application components are created through the implementation of the inbound interface and the wiring between the RedMaintenance inbound interface and the targeted application component. Follow these steps to create the RedMaintenanceInboundInterface:

1. Switch to the **Business Integration** perspective.
2. In the Business Integration view, right-click **Interface**.
3. In the context menu, select **New** → **Interface**.
4. Enter RedMaintenanceInboundInterface in the name field of the New Interface Wizard.

In the interface editor, create the operations as shown in Figure 12-4.

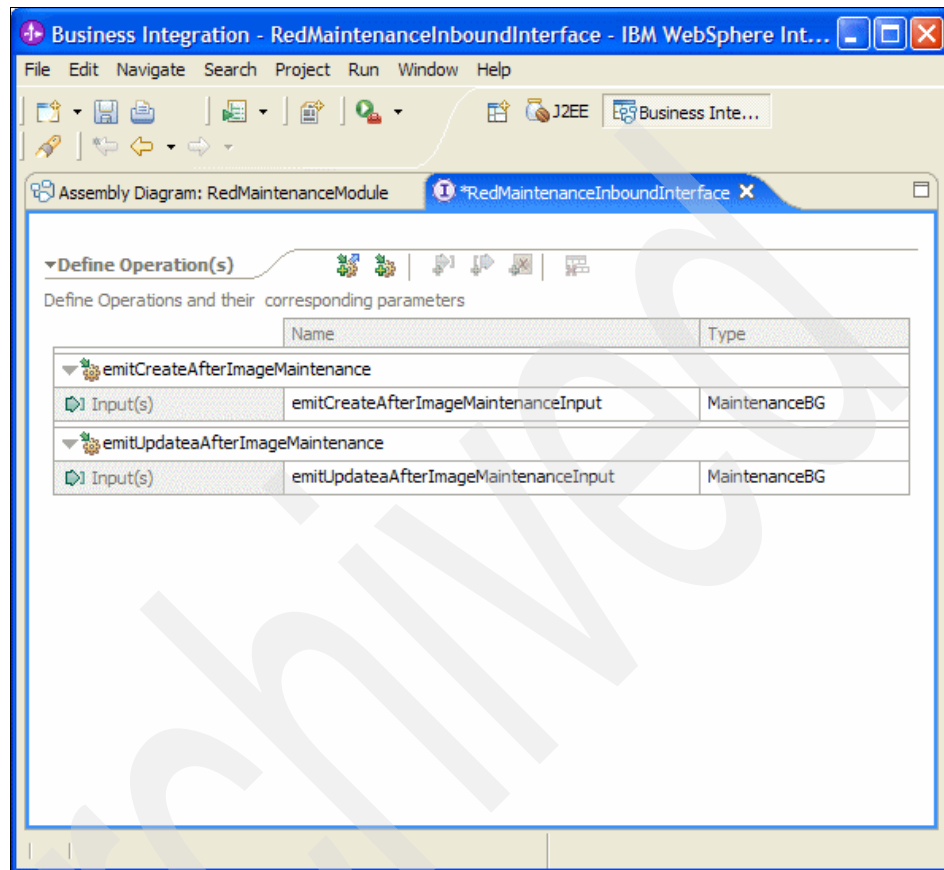


Figure 12-4 RedMaintenance Inbound interface

Note: For steps to create operations in an interface, see HelloWorld sample adapter in 8.5, “Importing a sample adapter into your development environment” on page 215.

12.9.3 Create the SCA EIS export file

Similar to the outbound processing described in the 11.7.6, “Create SCA EIS service import file” on page 317, the SCA EIS service export file specifies the mapping between the definition of inbound SCA operation in order that other application components can subscribe and receive business objects from the

resource adapter. It also binds the external EIS service to the SCA module. To create this file follow these steps:

1. Switch to **Business Integration** perspective.
1. On the menu bar select **Window** → **Show view** → **Physical Resources** to switch to **Physical Resources** view.
2. Create a new a simple file name `RedMaintenanceInboundInterface.export` file under `RedMaintenanceModule` folder.
3. Copy the content from Example 12-14 into this file.
4. **Save** this file.

Example 12-14 RedMaintenanceInboundInterface.export

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:export xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:eis="http://www.ibm.com/xmlns/prod/websphere/scdl/eis/6.0.0"
xmlns:ns1="http://RedMaintenanceModule/RedMaintenanceInboundInterface"
xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/6.0.0"
xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/6.0.0"
displayName="RedMaintenanceInboundInterface"
name="RedMaintenanceInboundInterface" target="SCATestComponent">
  <interfaces>
    <interface xsi:type="wsdl:WSDLPortType"
portType="ns1:RedMaintenanceInboundInterface">
      <method name="emitCreateAfterImageMaintenance"/>
      <method name="emitUpdateAfterImageMaintenance"/>
    </interface>
  </interfaces>
  <esbBinding xsi:type="eis:EISExportBinding"
dataBindingType="com.ibm.j2ca.extension.emd.runtime.WBIDataBindingImpl">
    <resourceAdapter name="RedMaintenanceModuleApp.RedMaintenance"
type="com.ibm.itso.sab511.RMResourceAdapter">
      <properties/>
    </resourceAdapter>
    <connection type="com.ibm.itso.sab511.inbound.RMActivationSpec"
selectorType="com.ibm.j2ca.extension.emd.runtime.WBIFunctionSelectorImpl">
      <properties>
        <BONamespace>http://www.ibm.com/xmlns/prod/wbi/j2ca/redmaintenance</BONamespace>
      </properties>
    </connection>
    <methodBinding method="emitCreateAfterImageMaintenance"
nativeMethod="emitCreateAfterImageMaintenance"/>
  </esbBinding>
</scdl:export>
```

```
<methodBinding method="emitUpdateAfterImageMaintenance"  
nativeMethod="emitUpdateAfterImageMaintenance"/>  
</esbBinding>  
</scdl:export>
```

Note: When this file is saved, a compilation problem appears because the target component (SCATestComponent) has not been created yet. Ignore this message because this error is corrected with the next step.

Also note that all these files described above, the business object files, the ASI files and the export file, are created automatically when ESD is used.

12.9.4 Create the SCATestComponent

For inbound processing, the inbound business object is delivered to message end points and components in WebSphere Process Server. To test inbound, we create a test target component.

The steps to create this target component are:

1. In the Business Integration Perspective, open the **Assembly Diagram** of the RedMaintenance module and select and click the Component (with no

implementation type) from the toolbar located in the right side of the Assembly Diagram window, as it is shown in Figure 12-5.

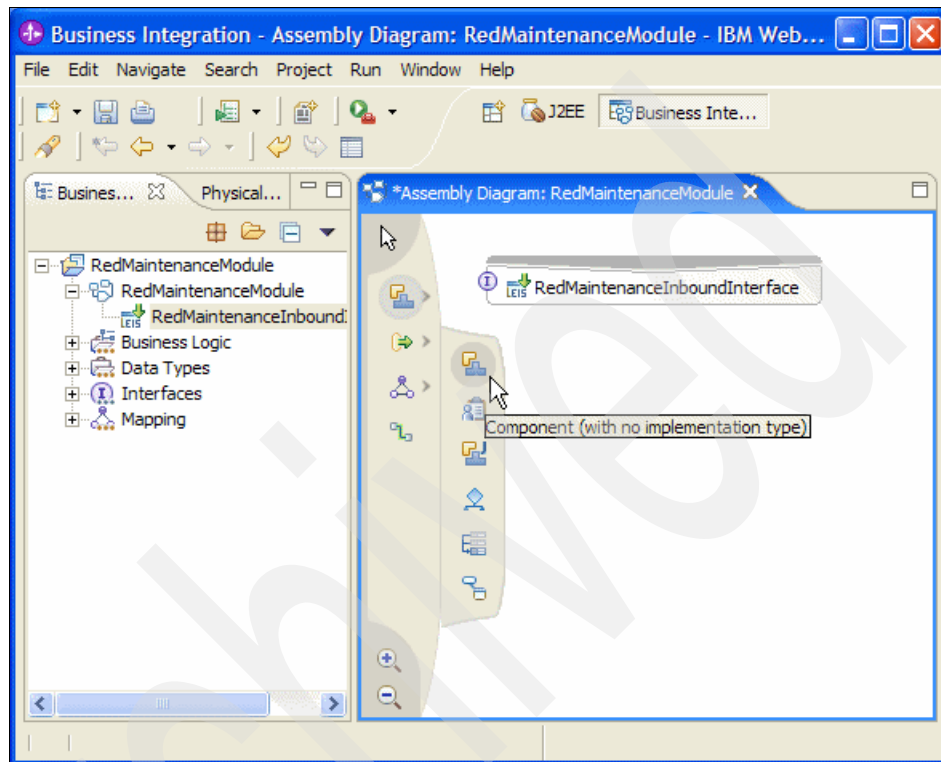


Figure 12-5 Creating a Component with no implementation type

- Click in an empty space in the assembly diagram, change the name of the new component to SCATestComponent, and wire the Inbound interface with the component as shown in Figure 12-6.

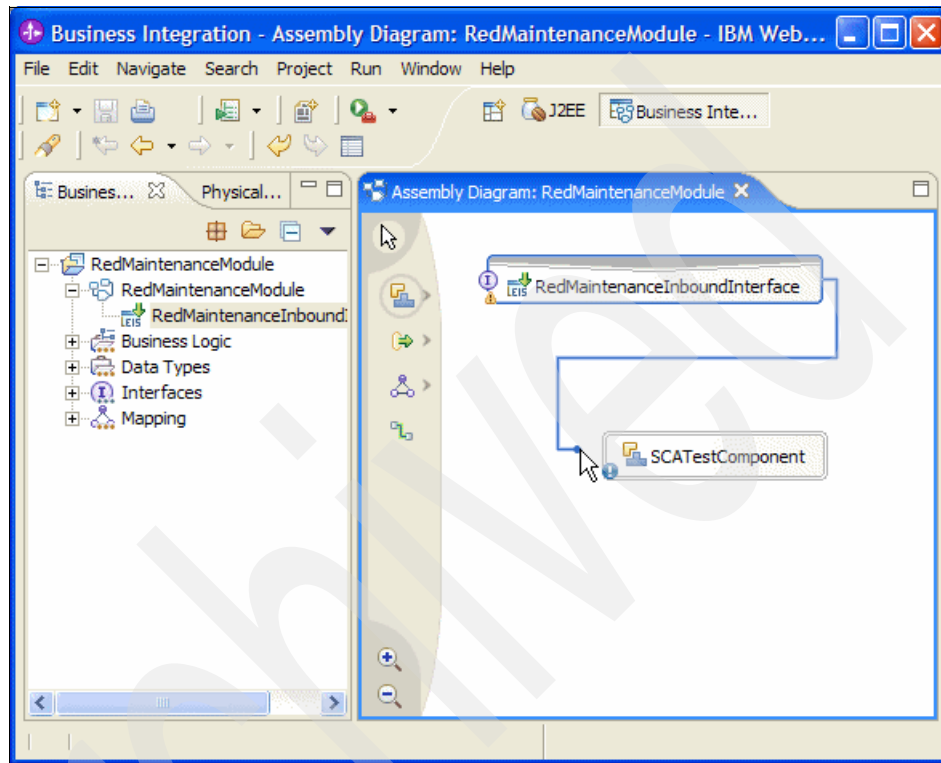


Figure 12-6 Wiring the Inbound interface with the SCATestComponent

- The next step is to generate the implementation of the new SCATestComponent. In our project, we generate a Java implementation using the context menu on the SCATestComponent and selecting **Generate Implementation** → **Java**.
- When the Generate Implementation dialog box appears, in the package name of the class define a new package with the name `com.ibm.itso.sab511.testzca`.
- Be sure that the code generated by the WebSphere Integration Developer is similar to the code listed in Example 12-15.

Example 12-15 SCATestComponentImpl class

```
package com.ibm.itso.sab511.testzca;  
  
import commonj.sdo.DataObject;
```

```

import com.ibm.websphere.sca.ServiceManager;

public class SCATestComponentImpl {
    /**
     * Default constructor.
     */
    public SCATestComponentImpl() {
        super();
    }

    /**
     * Return a reference to the component service instance for this
     implementation
     * class. This method should be used when passing this service to a partner
     reference
     * or if you want to invoke this component service asynchronously.
     *
     * @generated (com.ibm.wbit.java)
     */
    private Object getMyService() {
        return (Object) ServiceManager.INSTANCE.locateService("self");
    }

    /**
     * Method generated to support implementation of operation
     "emitCreateAfterImageMaintenance" defined for WSDL port type
     * named "interface.RedMaintenanceInboundInterface".
     *
     * The presence of commonj.sdo.DataObject as the return type and/or as a
     parameter
     * type conveys that its a complex type. Please refer to the WSDL
     Definition for more information
     * on the type of input, output and fault(s).
     */
    public void emitCreateAfterImageMaintenance(
        DataObject emitCreateAfterImageMaintenanceInput) {
        //TODO Needs to be implemented.
    }

    /**
     * Method generated to support implementation of operation
     "emitUpdateaAfterImageMaintenance" defined for WSDL port type
     * named "interface.RedMaintenanceInboundInterface".
     *
     * The presence of commonj.sdo.DataObject as the return type and/or as a
     parameter
     * type conveys that its a complex type. Please refer to the WSDL
     Definition for more information
     * on the type of input, output and fault(s).

```

```

        */
        public void emitUpdateaAfterImageMaintenance(
            DataObject emitUpdateaAfterImageMaintenanceInput) {
            //TODO Needs to be implemented.
        }
    }
}

```

6. Put some code in the SCATestComplmpl methods to test that they are called properly during the inbound processing test.

Now the inbound project is ready to be exported, deployed and tested.

12.9.5 Exporting the adapter

To export the adapter follow the instructions in the 11.7.7, “Exporting the adapter” on page 318 of this book.

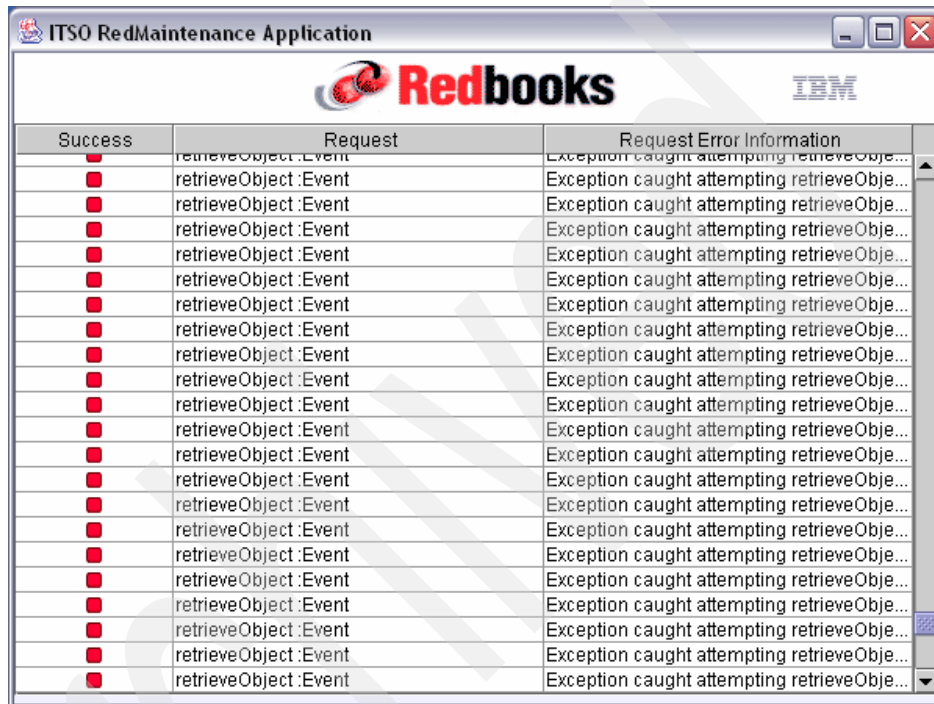
12.10 Test inbound operations

In this section we explain how to test RedMaintenance’s inbound processing. The following steps are performed for inbound testing:

1. Start **RedMaintenance server**.
2. Start **WebSphere Process Server** from WebSphere Integration Developer.
3. Install **RedMaintenance module** application to WebSphere Integration Developer’s instance of WebSphere Process Server.
4. Import a sample stand-alone RedMaintenance inbound test client to our adapter workspace. This sample client is used to create, update Maintenance records in RedMaintenance.
5. Set up integration test client in WebSphere Integration Developer to monitor inbound business object delivered by the adapter to WebSphere Process Server.
6. Run stand-alone **RedMaintenance inbound test client** to create, update Maintenance records in RedMaintenance.
7. Observe the business object delivered to WebSphere Process Server from integration test client.

The instruction for steps 1 to 3 are the same as the steps in 11.8, “Test outbound operations” on page 320.

When you start the RedMaintenanceModule in WebSphere Process Server, you can see events occurring on RedMaintenance application's user interface. See Figure 12-7. These events shows the RedMaintenance adapter is polling RedMaintenance application's event store. The exceptions indicate there are no events to retrieve.



Success	Request	Request Error Information
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...
■	retrieveObject : Event	Exception caught attempting retrieveObje...

Figure 12-7 Events indicating RedMaintenance adapter is polling.

12.10.1 Sample stand-alone RedMaintenance test client

RedMaintenance application has a built-in event mechanism that records Maintenance entity's create and update events to a event table. To test RedMaintenance adapter's inbound processing, we have created an stand-alone RedMaintenance client that can create a new Maintenance record as well as updating an existing Maintenance record. Follow these steps to import this sample stand-alone RedMaintenance client:

1. Create a new Java project in WebSphere Integration Developer. The new project is created in the same workspace as our adapter workspace.
2. On the New Java Project dialog enter ExternalRMCClient in the Project name field.

3. Click **Finish** to create the project.
4. Right-click the newly created project and select **Properties** to open the Properties for ExternalRMClient dialog box.
5. Select **Java Build Path**.
6. Under the **Libraries** tab, add RedMaintenance's `common.jar` to the build path.
7. Import the file `SG246387\RedMaintenance\policy.txt` into the project.
8. Import the file `SG246387\RedMaintenance\client.jar` into the project. When you import this file, make sure you select **Zip file** on the Import dialog box. When you are finished, the project appears as shown in Figure 12-8.

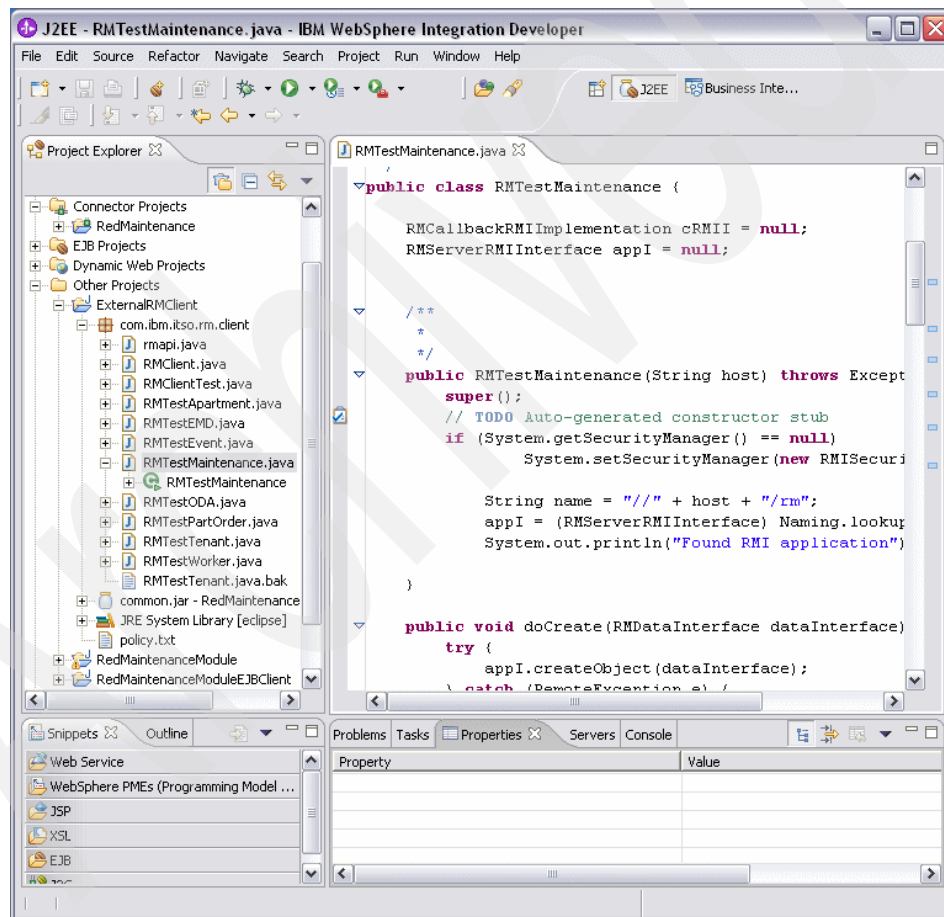


Figure 12-8 Import RedMaintenance client

We use `RMTestMaintenance.java` class to create and update the Maintenance entity in the RedMaintenance application.

12.10.2 Setup integration test client

WebSphere Integration Developer provides an integration test client. From the test client we can monitor the inbound business objects that are delivered to WebSphere Process Server by the RedMaintenance adapter. The following steps show you how to attach RedMaintenanceModule to the test client:

1. Switch to **Business Integration** perspective.
2. Right-click **RedMaintenanceModule** → **test** → **attach** to attach RedMaintenanceModule to the integration test client. The integration test client is show in Figure 12-9.

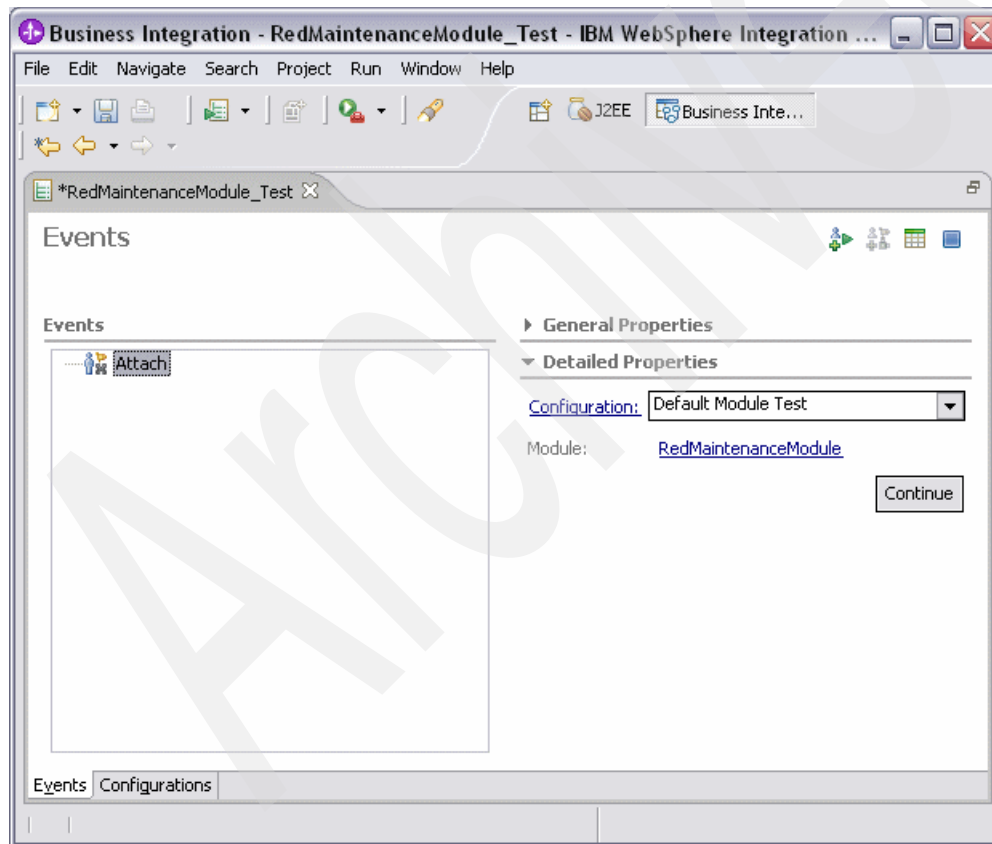


Figure 12-9 Integration test client.

12.10.3 Run stand-alone RedMaintenance test client

To test inbound processing, we use the stand-alone RedMaintenance test client to create and update Maintenance records in RedMaintenance application. Follow these steps to run this test client:

1. Right-click **RMTestMaintenance.java** class. See Figure 12-8 on page 387.
2. Click **Run** to open the Run dialog box.
3. Select **RMTestMaintenance**.
4. Enter `-Djava.security.policy=policy.txt` in the VM arguments section under the Arguments tab. See Figure 12-10.

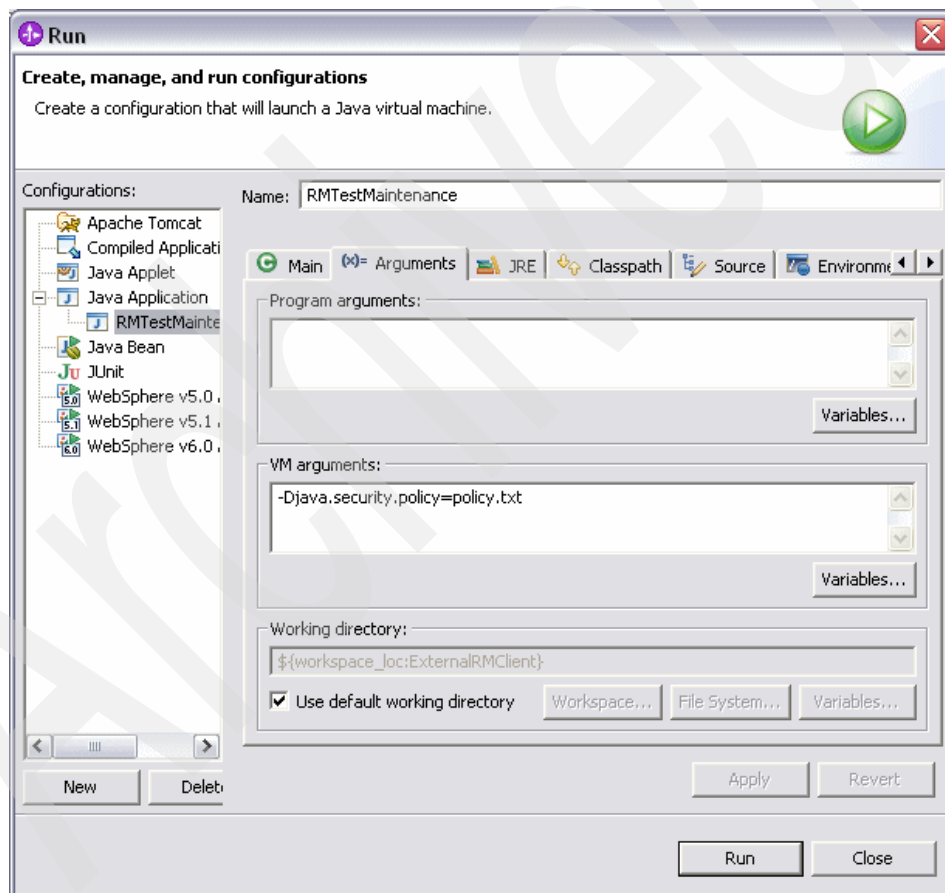
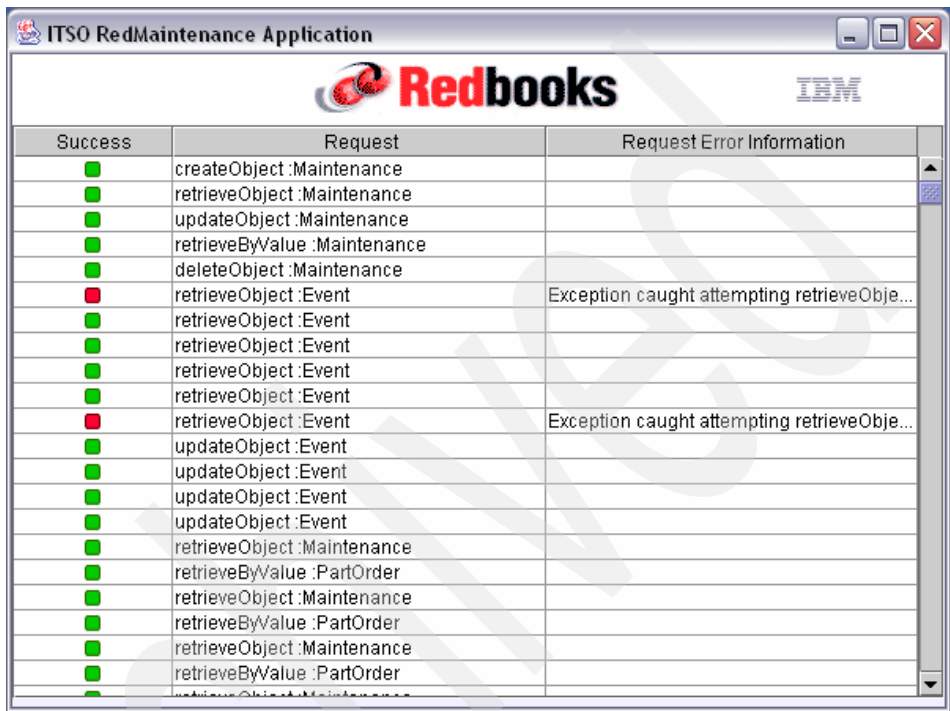


Figure 12-10 Run dialog.

5. Click **Run** to create and update Maintenance records.

12.10.4 Observe the effects of these inbound events

Figure 12-11 shows RedMaintenance application's Maintenance entity being created and updated.



Success	Request	Request Error Information
■	createObject :Maintenance	
■	retrieveObject :Maintenance	
■	updateObject :Maintenance	
■	retrieveByValue :Maintenance	
■	deleteObject :Maintenance	
■	retrieveObject :Event	Exception caught attempting retrieveObje...
■	retrieveObject :Event	
■	retrieveObject :Event	
■	retrieveObject :Event	
■	retrieveObject :Event	
■	retrieveObject :Event	Exception caught attempting retrieveObje...
■	updateObject :Event	
■	updateObject :Event	
■	updateObject :Event	
■	updateObject :Event	
■	retrieveObject :Maintenance	
■	retrieveByValue :PartOrder	
■	retrieveObject :Maintenance	
■	retrieveByValue :PartOrder	
■	retrieveObject :Maintenance	
■	retrieveByValue :PartOrder	
■	retrieveObject :Maintenance	

Figure 12-11 Inbound events on RedMaintenance application

Figure 12-12 shows the RedMaintenance adapter delivered the Maintenance business graph to WebSphere Integration Developer's integration test client.

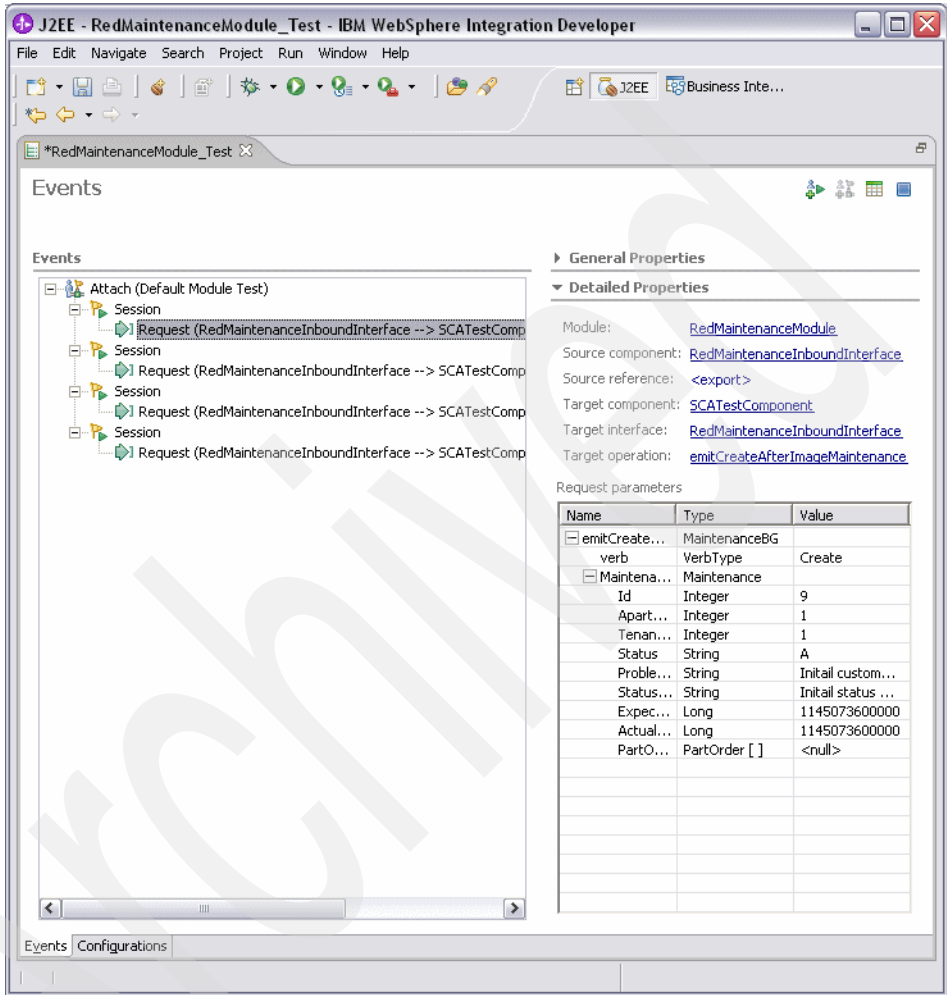


Figure 12-12 Inbound business objects in WebSphere Process Server

12.11 Configure logging and tracing levels for debugging

Chapter 7, “Exceptions, logging, and tracing” on page 147 explains logging and tracing. Our code example shows you where you can place logging and trace code. How to configure this is discussed in 11.9, “Configure logging and tracing levels for debugging” on page 336.

Implementing Enterprise Metadata Discovery

This chapter explains how to implement Enterprise Metadata Discovery (EMD) for RedMaintenance adapter. We provide details on how to extend Adapter Foundation Classes default EMD implementation to speed up adapter EMD development.

Creating a custom resource adapter that support EMD involves the following steps:

- ▶ Design what kind of metadata in EIS needs to be discovered and generated by EMD.
- ▶ Using WebSphere Adapter Toolkit to generate the code stubs for the resource adapter EMD.
- ▶ Create EMD deployment descriptor.
- ▶ Create schema file to constrain application-specific information annotations in the generated business object definition.
- ▶ Understand utility APIs provided by EMD Tooling.
- ▶ Implement EMD stub classes generated by WebSphere Adapter Toolkit.
- ▶ Testing the EMD implementation.

13.1 Analyze EIS metadata before EMD implementation

Different EISs have different ways to represent data. Before implementing EMD, we need to ask ourselves the following questions:

- ▶ What kind of data in EIS system is exposed?
- ▶ What is the data format in EIS?
- ▶ How do we represent the EIS data as business object?

In a database system, most data are stored in the database tables. So, we could simply use one business object to represent one table. As from EMD's point of view, it needs to generate one business object definition XSD file to represent one database table in the database.

In a file system, data are stored as files. So, we use one business object definition XSD file to represent a file., This business object definition contains XML elements to represent file name, directory path, read only attribute and so on.

The IBM Red Maintenance Adapter connects to the RedMaintenance server. The RedMaintenance server's data are stored as tables. We could use the method that one business object represents one table for RedMaintenance server.

Here we can answer the questions:

- ▶ What kind of data in an EIS system is exposed?
All data stored in RedMaintenance server
- ▶ What is the data format in EIS?
Database table
- ▶ How do we represent the EIS data as a business object?
One business object represents one table

13.2 Use WebSphere Adapter Toolkit to generate EMD stub classes

In the previous two chapters we explain how to use WebSphere Adapter Toolkit to create the stub classes for RedMaintenance adapter. We also explain how to implement the stub class to support adapter inbound and outbound processing. In this chapter we explain how add EMD support for RedMaintenance adapter. If you are continuing from the previous chapter, skip the next section and go directly to Section 13.2.2, "Using an existing adapter project" on page 399.

Otherwise you can use WebSphere Adapter Toolkit to create a new adapter project, described in 13.2.1, “Start a new adapter project” on page 395.

13.2.1 Start a new adapter project

To start a new adapter project, follow these steps:

1. In the WebSphere integration developer with WebSphere Adapter Toolkit, Select:
 - a. **File** → **New project** → **Adapter Toolkit** → **J2C resource Adapter Project** and click **Next**. This launches the wizard to create a J2C resource adapter project. See Figure 13-1.

Note: If you cannot find the Adapter Toolkit item, ensure that the Show All Wizards option at the bottom of the panel is checked.

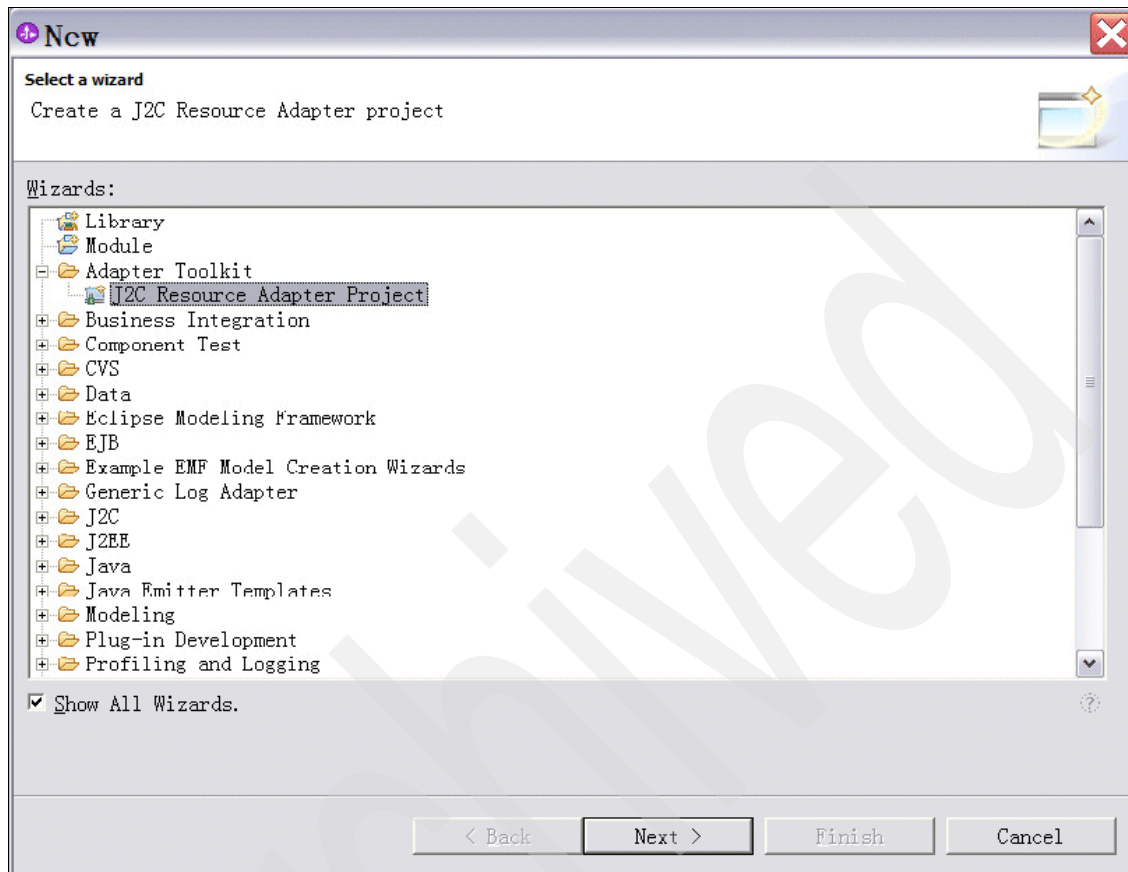


Figure 13-1 Create a new adapter project

2. In the new project dialog box, provide the name for the project and select the other project-related information such as **JCA Version** and **Target Server**, and click **Next** to proceed. WebSphere Adapter Toolkit V6 only supports creating JCA 1.5 based Resource Adapters with WebSphere Process Server 6 as the target server. See Figure 13-2.

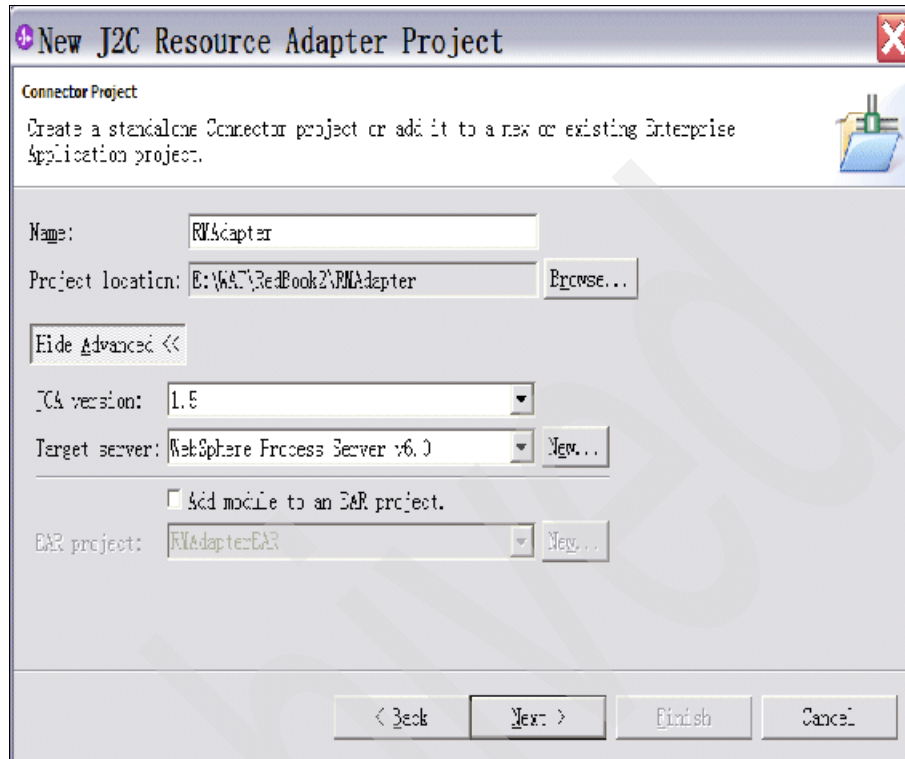


Figure 13-2 New project dialog

3. In the Adapter Properties dialog box, enter the adapter-specific details, which include adapter name, package name, and class name. Click **Next** to proceed.
4. In the Generation Options dialog box, first select **IBM WebSphere Resource Adapter** as the adapter type. Only after selecting IBM WebSphere Resource Adapter, can we create EMD-related classes. In the middle of this dialog box, select the features (classes) you want to generate. To create EMD, check **Generate Enterprise Metadata Discovery classes**. See Figure 13-3 on page 398.

In Figure 13-3 on page 398, as we have mentioned in the EMD theory chapter, EMD usually use code from outbound implementation to get a connection to the EIS. So, it is better to check **Generate Outbound Adapter classes** at the same time.

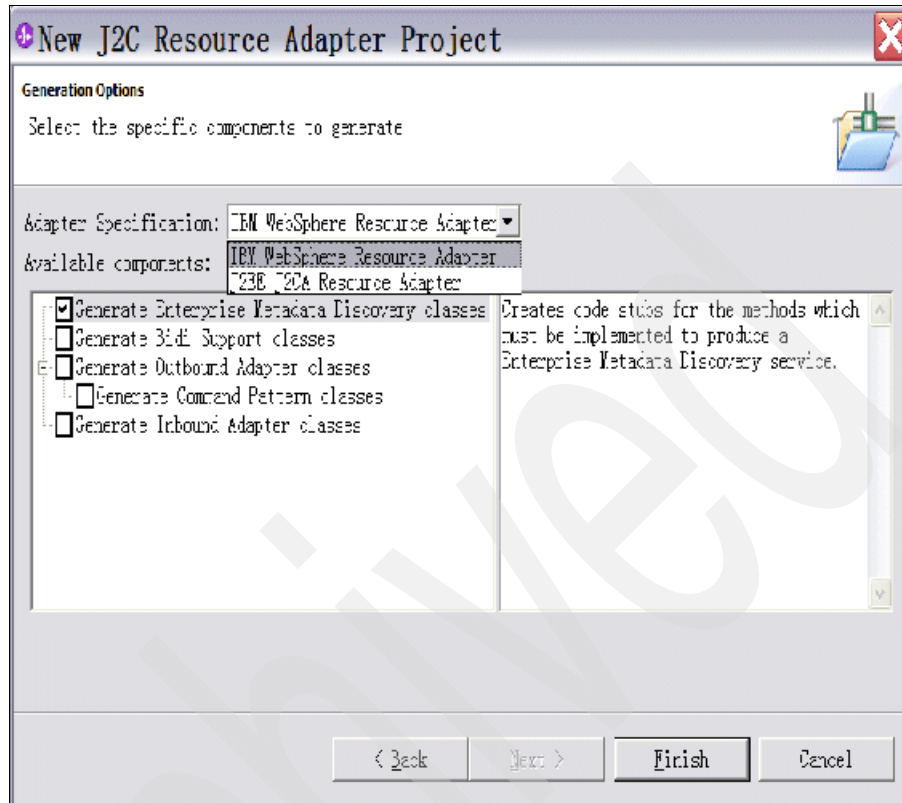


Figure 13-3 Generation options dialog

Now all the required code stubs for EMD module are generated. Later in this chapter, we discuss how to implement these generated classes in detail.

Figure 13-4 shows the EMD stub classes generated by WebSphere Adapter Toolkit.

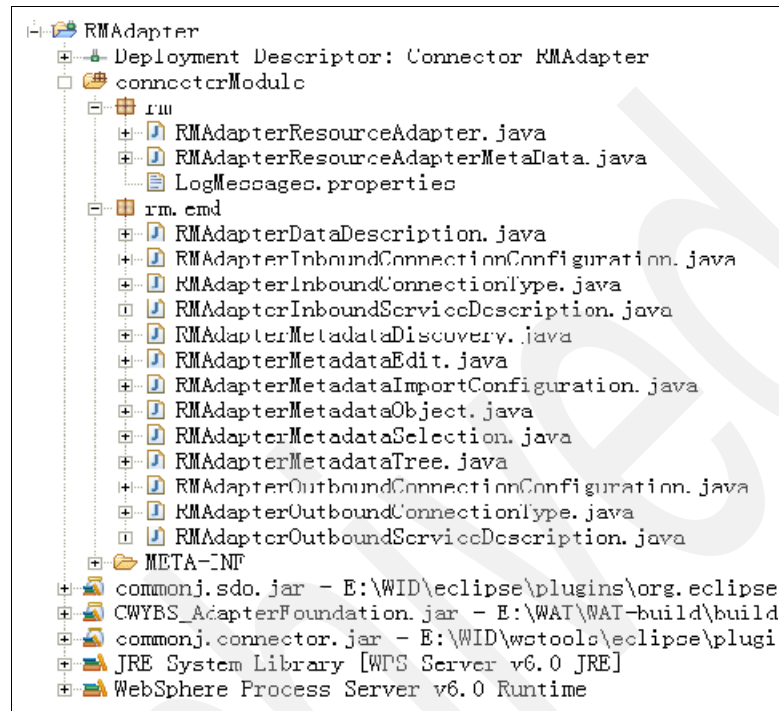


Figure 13-4 EMD stub classes generated by WebSphere Adapter Toolkit

EMD uses some outbound classes for connecting to EIS, so it is better to implement outbound classes first. In fact, if you plan to create a JCA adapter with outbound, inbound and EMD support, it better to code stubs for inbound, outbound and EMD when you are creating the adapter project.

13.2.2 Using an existing adapter project

If the adapter project has already been created, follow these steps to add the inbound stub classes:

1. Open the **RedMaintenance** project, and edit the Resource Adapter Deployment Descriptor using the Deployment Descriptor Editor tool.
2. In the Overview panel of the Deployment Descriptor, look for the **Component Addition** option and click **Add**.

Figure 13-5 gives an example of that action.

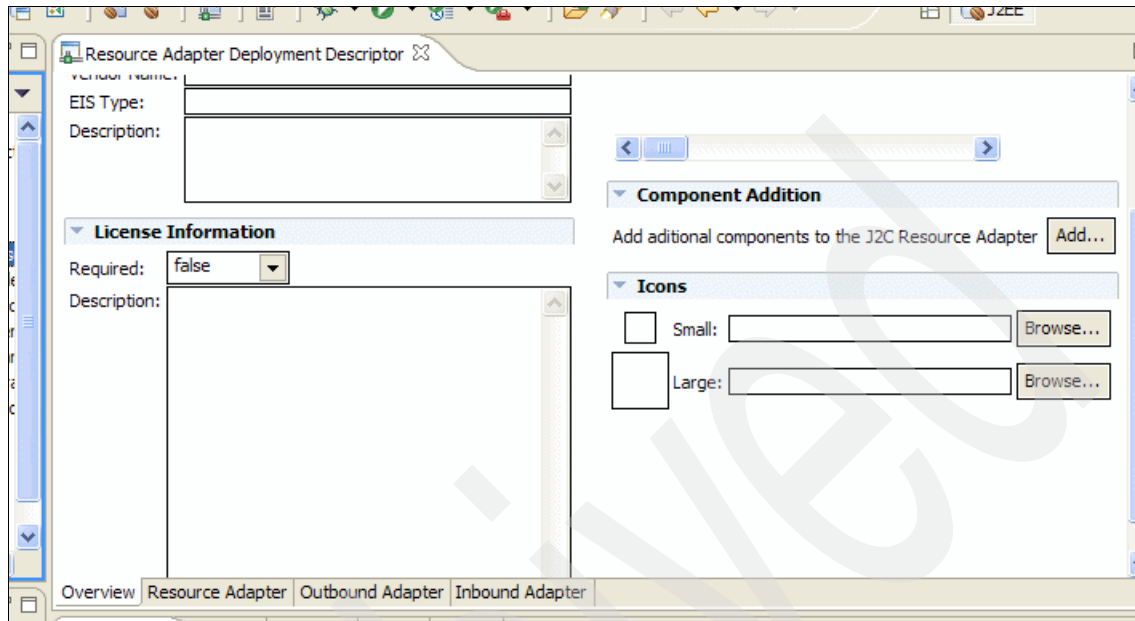


Figure 13-5 Adding inbound processing to an existent J2C adapter project

3. In the Add Component dialog box, select **Generate Enterprise Metadata Discovery classes** and click **Finish**.
4. The generated inbound classes now can be viewed in the J2EE perspective, under the `com.ibm.itso.sab511.inbound` package.

13.2.3 Create the deployment descriptor for EMD

Each adapter EMD must have a deployment descriptor file named `discovery-service.xml`. This file must be created in the folder `/connectorModule/META-INF/`. The ESD tools loads the configurations from this file to discover the EMD name, the EMD main classes, and so on.

This file is not automatically generated by WebSphere Adapter Toolkit, we need to create it manually.

Example 13-1 on page 401 is a sample `discovery-service.xml` file, in which we need to fill the following information:

- ▶ EMD display name and description
- ▶ EMD vendor name and version
- ▶ Metadata Discovery class and Metadata Editor class

These are two main entrance classes that ESD Tools calls. We need to set full package name for the two classes:

- ▶ Application-specific information (ASI) schema URI
- ▶ Application-specific information file location

Example 13-1 EMD deployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<emd:discoveryService xmlns:emd="commonj.connector"
  xmlns:j2ee="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <j2ee:description>RedMaintenance EMD Adapter</j2ee:description>
  <j2ee:display-name>RedMaintenance EMD Adapter</j2ee:display-name>
  <emd:vendor-name xsi:type="j2ee:xsdStringType">IBM</emd:vendor-name>
  <emd:version xsi:type="j2ee:xsdStringType">1.0.0</emd:version>
  <emd:spec-version>1.0</emd:spec-version>
  <emd:discoveryService-class
    xsi:type="j2ee:fully-qualified-classType">
    com.ibm.itso.sab511.emd.RMMetadataDiscovery
  </emd:discoveryService-class>
  <emd:metadataEdit-class xsi:type="j2ee:fully-qualified-classType">
    com.ibm.itso.sab511.emd.RMMetadataEdit
  </emd:metadataEdit-class>
  <emd:application-specific-schema>
    <j2ee:description>RedMaintenance ASI schema</j2ee:description>
    <j2ee:display-name>RedMaintenance ASI schema</j2ee:display-name>
    <emd:asINSURI>
      http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
    </emd:asINSURI>
    <emd:asiSchemaLocation>RedMaintenanceASI.xsd</emd:asiSchemaLocation>
  </emd:application-specific-schema>
</emd:discoveryService>
```

Here, we just need to decide the schema file name and its URI. In the next step, we describe what the ASI schema file is and how to create it.

13.2.4 Application-specific information schema

The application-specific information schema of the adapter is defined in the ASI file. This allows the business object definition to include additional metadata information. The additional metadata information can be added to the business object definition in the form of XML annotations. The purpose of the ASI XSD file is to constrain the metadata annotation information that is added to the business object definition. The ASI metadata information in the business object definition is utilized by the adapter to process inbound and outbound operations.

Three kinds of metadata information can be set:

- ▶ Business object-level metadata, which contains information for the whole business object (BO).
- ▶ Verb-level metadata, which contains information for each verb.
- ▶ Attribute-level metadata, which contains information for the individual property.

A sample of the ASI file is given in Example 13-2 for the Red Maintenance Adapter.

Different EIS have different metadata. For example, in a database system, one business object usually represents one table, and each attribute in the BO represents one field in the table. Then, the BO-level metadata could be the table name and so on. The verb-level metadata could be the method name and so on. The attribute level metadata could be `IsPrimaryKeyField` and so on.

Example 13-2 is the ASI schema file for RedMaintenance Adapter EMD, in which we set all three levels of metadata. Refer to Chapter 3, “Service Data Objects” on page 53 for more information about the ASI.

Example 13-2 Application-specific information schema

```
<xs:schema
  targetNamespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/me
  tadata"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"

  xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
  "
  elementFormDefault="qualified">
  <xs:import
    namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/base/metadata"
    schemaLocation="BaseAdapterMetadata.xsd" />

  <!-- ASI supported at BO-level in business object definitions -->
  <xs:complexType name="BusinessObjectTypeMetadata">
    <xs:sequence minOccurs="0" maxOccurs="1">
      <xs:element name="ObjectName" type="xs:string" />
      <xs:element name="SupportedVerbs" minOccurs="0"
        maxOccurs="1">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Create"
              type="asi:VerbTypeMetadata" />
            <xs:element name="Update"
```

```

        type="asi:VerbTypeMetadata" />
        <xs:element name="Delete"
            type="asi:VerbTypeMetadata" />
        <xs:element name="Retrieve"
            type="asi:VerbTypeMetadata" />
    </xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>

<!-- ASI supported per top-level verb -->
<xs:complexType name="VerbTypeMetadata">
    <xs:sequence minOccurs="0" maxOccurs="1">
        <xs:element name="MethodName" type="xs:string" />
    </xs:sequence>
</xs:complexType>

<!-- ASI supported at property-level in business object definitions -->
<xs:complexType name="PropertyTypeMetadata">
    <xs:sequence minOccurs="0" maxOccurs="1">
        <xs:element name="PrimaryKey" type="xs:boolean" />
        <xs:element name="ForeignKeyB0Ref" type="xs:string" />
        <xs:element name="FieldName" type="xs:string" />
        <xs:element name="FieldType" type="xs:string" />
    </xs:sequence>
</xs:complexType>
</xs:schema>

```

13.2.5 Understand utility APIs provided by EMD tooling

The utility APIs provided by the adapter foundation classes and ESD tools perform some common functions like logging, monitoring progress of ESD. Using these utilities in our code expedites our development progress.

There are three main utility APIs:

- ▶ Logging and Tracing

Used for logging and tracing:

- To get a logger handle:

```
logUtils = WBIMetadataDiscovery.getLogUtils();
```

- To use the logger:

```
logUtils.log(Level.SEVERE, BUNDLE Reference, CLASS_NAME,
"message", "log number", new Object[] {});
```

► Property Group

To represent the properties, the following classes can be used to wrap and process data:

- WBIPropertyGroupImpl.class
- WBISingleValuedPropertyImpl.class
- WBIMultiValuedPropertyImpl.class
- Also, the EMDUtil class can be used to copy the values between properties:

```
EMDUtil.copyValues(propertyGroup1, propertyGroup2);
```

► Progress Monitor

In order to notify the user about the current status of the ESD Tools, a progress monitor can be used to monitor the progress of the service discovery as shown in Example 13-3.

Example 13-3 Using EMD utility APIs

```
WBIMetadataConnectionImpl.getToolContext().getProgressMonitor().setMaximum(100)
;
WBIMetadataConnectionImpl.getToolContext().getProgressMonitor().setMinimum(0);
WBIMetadataConnectionImpl.getToolContext().getProgressMonitor().setProgress(50)
;
WBIMetadataConnectionImpl.getToolContext().getProgressMonitor().setNote("message");
```

13.2.6 Implement EMD stub classes

The stub classes generated by WebSphere Adapter Toolkit are extension of EMD classes provided by Adapter Foundation Classes. Adapter Foundation Classes provides default EMD implementation. This greatly helps us to speed up EMD development. We can leverage the default EMD implementation that handles common business logic for EMDs and only provide our EIS specific implementation for our EMD.

In our example, RedMaintenance resource adapter EMD component performs the following tasks:

1. Connect to RedMaintenance server.
2. Retrieve metadata from it.
3. Allow the user to prune the metadata tree.
4. Generate the service interfaces.
5. Generate service import.
6. Generate service export.
7. Generate business object definitions.

The rest of this section introduces EMD classes and methods one by one to describe the function of each class or method and how to implement it.

RMMetadataDiscovery

This class provides methods that work for discovering a service. By calling the corresponding methods, EMD Tooling gets the adapter type summaries, metadata tree instance and so on. During generating service interface, the createServiceDescription method is called by EMD Tooling. Typically, we need to implement the following methods:

- ▶ `public AdapterTypeSummary[] getAdapterTypeSummaries()`
This method returns the vector of supported adapter connection types. For RedMaintenance, three types are returned. They are outbound connection for metadata discovery, outbound connection for runtime, and inbound connection for runtime.
- ▶ `public MetadataTree getMetadataTree(MetadataConnection conn)`
This method returns a MetadataTree instance, which contains the metadata object structures that represent the data structure in EIS. This tree structure is seen by users who select these objects.
- ▶ `public ServiceDescription createServiceDescription(MetadataSelection importSelection)`
This method is the main entrance of importing service description and BOs. After all the required information is collected from users, ESD Tools calls this method to create service description, function description and business objects. Later, the above data structures is output as the corresponding import/export file, wsdl interface file and business object schema file (*.xsd).
- ▶ `public void setToolContext(ToolContext toolContext)`
This method is to set the logUtil and those resources that ESD Tools provide for the later process.

RMMetadataEdit

ESD tools (WID) calls the methods in the metadata edit class to edit the metadata. The service interface and business object generated by EMD are stored as files in the project of WID. WID as a tool provide editor framework for users to edit those types of files. For EMD, we need to provide the details information related to our resource adapter in order to let WID know that which properties in the service interface are required, which properties are valid and so on.

- ▶ `public RMMetadataEdit()`
The constructor creates an instance of WBIAdapterTypeImpl for editing the metadata.

- ▶ `public OutboundConnectionType getOutboundConnectionType(String arg0)`
This method returns the outbound connection type for runtime. The connection types are stored in the `WBIAdapterTypeImpl` instance.
- ▶ `public InboundConnectionType getInboundConnectionType(String arg0)`
This method returns the inbound connection type for runtime. The connection types are stored in the `WBIAdapterTypeImpl` instance.

RMOutboundConnectionType

This class represents the outbound connection type to RedMaintenance server. The outbound connection type has the connection configuration which contains the connection properties, and has its method, `openMetadtaConnection`, to get the real connection to RedMaintenance server. We need to implement the following methods for this class:

- ▶ `public OutboundConnectionConfiguration createOutboundConnectionConfiguration()`
This method is called by EMD Tools to create the outbound connection configuration, which contains the properties of connecting to EIS both for metadata discovery and for the runtime requirement stored in the *.import file.
- ▶ `public MetadataConnection openMetadataConnection(OutboundConnectionConfiguration conf)`
This method returns a `MetadataConnection`, in which the physical EIS connection is initialized for retrieving metadata.

RMOutboundConnectionConfiguration

This class represents the outbound connection configurations to RedMaintenance server. EMD Tools call the method `createUnifiedProperties` to get the connection properties. For outbound, there are two types of connection properties. One is for metadata discovery, which is seen on the first panel of EMD Tools. After user input the values, those properties are used to connect to RedMaintenance server for retrieving metadata. The other is for runtime, which is seen on the final editable panel of EMD Tools. After users input values they want, EMD Tools stored those properties in the *.import file.

The method `createUnifiedProperties` create two types of properties for outbound, but you may notice that it does not have input parameters. How could it do?

The answer is that the `OutboundConnectionConfiguration` instance is initialized by `OutboundConnectionType`. The latter passes itself to the former, which means the `OutboundConnectionConfiguration` instance has the reference to the `OutboundConnectionType` instance. And the `OutboundConnectionType` instance has the method of `isSupportedInMetadataService`. If this method returns true,

createUnifiedProperties methods creates properties for metadata discovery; otherwise it creates properties for runtime:

► public PropertyGroup createUnifiedProperties()

This method is for creating property group both for retrieving metadata from EIS and for adapter runtime. Properties like connect string, user name, and password need to be put in the property group. A sample is shown in Example 13-4.

Example 13-4 Sample createUnifiedProperties method implementation

```
public commonj.connector.metadata.discovery.properties.PropertyGroup
createUnifiedProperties() {
    WBIPropertyGroupImpl propGroup = null;
    try {
        //Create a property group for Metadata
        if
(this.getOutboundConnectionType().isSupportedInMetadataService()) {
            propGroup = new WBIPropertyGroupImpl("ConnectionProperties");
            propGroup.setDisplayName("ConnectionProperties");
            propGroup.setDescription("ConnectionProperties");

            // Add the host name/ip
            WBISingleValuedPropertyImpl hostProp = new
WBISingleValuedPropertyImpl("ServerURL", String.class);
            hostProp.setDescription("ServerURL");
            hostProp.setDisplayName("ServerURL");
            hostProp.setDefaultValue("localhost");
            hostProp.setRequired(true);
            propGroup.addProperty(hostProp);

            WBISingleValuedPropertyImpl rmiProp = new
WBISingleValuedPropertyImpl("RmiName", String.class);
            rmiProp.setDescription("RmiName");
            rmiProp.setDisplayName("RmiName");
            rmiProp.setDefaultValue("rm");
            rmiProp.setRequired(true);
            propGroup.addProperty(rmiProp);

            // Add the Prefix
            WBISingleValuedPropertyImpl prefixProp = new
WBISingleValuedPropertyImpl("Prefix", String.class);
            prefixProp.setDescription("Prefix");
            prefixProp.setDisplayName("Prefix");
            propGroup.addProperty(prefixProp);

        } else { //Create a property group for runtime

            propGroup = new WBIPropertyGroupImpl("OutboundProperties");
```

```

        propGroup.setDisplayName("OutboundProperties");
        propGroup.setDescription("OutboundProperties");

        WBIPropertyGroupImpl mcfPropGrp = (WBIPropertyGroupImpl)
createManagedConnectionFactoryProperties();
        if(mcfPropGrp!=null) propGroup.addProperty(mcfPropGrp);

        WBIPropertyGroupImpl raPropGrp = (WBIPropertyGroupImpl)
createResourceAdapterProperties();
        if(raPropGrp!=null) propGroup.addProperty(raPropGrp);
    }
    if (this.getOutboundConnectionType().isSupportedInMetadataService()
&& this.getAppliedProperties() != null) {
        EMDUtil.copyValues(getAppliedProperties(), propGroup);
    }
    } catch (Exception e) {
        WBIMetadataDiscoveryImpl.getLogUtils().log(Level.SEVERE,
LogUtilConstants.ADAPTER_RBUNDLE, "RMOutboundConnectionConfiguration",
"createUnifiedProperties", "1000", new Object[] { e.getMessage()
});//$NON-NLS-1$
        throw new RuntimeException(e.getMessage(), e);
    }

    return propGroup;
}

```

-
- ▶ **public PropertyGroup createManagedConnectionFactoryProperties()**
This method is for creating the properties that related to ManagedConnectionFactory class.
 - ▶ **public PropertyGroup createResourceAdapterProperties()**
This method is for creating the properties that related to the resource adapter class.

RMOutboundServiceDescription

This class stores the methods that used to generate the service interface and business objects for outbound:

- ▶ **public void setFunctionDescriptions(MetadataSelection selection)**
This method is called by RMMetadataDiscovery.createServiceDescription(). In this setFunctionDescriptions method, we get MetadataSelection as input, which contains all the metadata objects selected by users. For each metadata object, we need to set most properties here, like BO location, relative path, BO attributes, namespace, and so on.

Example 13-5 shows the sample.

Example 13-5 Sample setFunctionDescriptions method implementation

```
try {
    ArrayList functionDescriptions = new ArrayList();
    //Selection props will be used to get the relative path
    PropertyGroup selectionProps =
    WBIMetadataSelectionImpl.getAppliedSelectionProperties();
    MetadataImportConfiguration[] confArray = selection.getSelection();
    String namespaceValue = "";

    for (int i = 0; i < confArray.length; i++) {
        String location = ".";
        WBIMetadataImportConfigurationImpl spec = (WBIMetadataImportConfigurationImpl)
        confArray[i];
        WBIOutboundFunctionDescriptionImpl funcDesc;
        RMInteractionSpec iSpec;
        RMMetadataObject metadataObj = (RMMetadataObject) spec.getMetadataObject();
        WBISingleValuedPropertyImpl locationProp = (WBISingleValuedPropertyImpl)
        selectionProps.getProperty("RELATIVEPATH");
        if (locationProp.getValue() != null) {

            String locationPropValue = (String) locationProp.getValue();
            if (locationPropValue.trim().length() != 0) {

                location = location + locationPropValue;
            }
        }
        //If location does not end with '/' add it
        if (!location.endsWith("/")) {

            location = location + "/";
        }

        WBISingleValuedPropertyImpl namespace = (WBISingleValuedPropertyImpl)
        selectionProps.getProperty("NAMESPACE");
        if (locationProp.getValue() != null) {
            namespaceValue = (String) namespace.getValue();
            if (namespaceValue.trim().length() == 0) {
                namespaceValue = "http://www.ibm.com/xmlns/prod/wbi/j2ca/redmaintenance";
            }
        }
    }

    String operation = "Execute";
    this.setNamespace(namespaceValue);
    metadataObj.populateBO();
    funcDesc = new WBIOutboundFunctionDescriptionImpl();
    funcDesc.setName(operation.toLowerCase() + metadataObj.getBOName());
}
```

```

RMDataDescription dataDesc = new RMDataDescription();
dataDesc.setMetadataObject(metadataObj);
dataDesc.setName(getNamespace(), metadataObj.getMetadata().getComponentName());

dataDesc.setRelativePath(location);
dataDesc.populateSchemaDefinitions();
dataDesc.setName(BusinessObjectDefinition.convertNamespaceToUri(getNamespace()
+ "/" + metadataObj.getBOName().toLowerCase() + "BG".toLowerCase()),
metadataObj.getBOName() + "BG");
funcDesc.setInputDataDescription(dataDesc);
funcDesc.setOutputDataDescription(dataDesc);
iSpec = new RMInteractionSpec();
iSpec.setFunctionName(operation);
funcDesc.setInteractionSpec(iSpec);
funcDesc.setImportConfiguration(spec);
functionDescriptions.add(funcDesc);
}

FunctionDescription[] funcArray = new
FunctionDescription[functionDescriptions.size()];
functionDescriptions.toArray(funcArray);
super.setFunctionDescriptions(funcArray);
} catch (Exception e) {
throw new MetadataException(e.getMessage(), e);
}
}

```

RMInboundConnectionType

This class represents the inbound connection type to RedMaintenance server. The inbound connection type has the connection configuration which contains the connection properties. We need to implement the following methods for this class:

- ▶ public InboundConnectionConfiguration
createInboundConnectionConfiguration()

This method is called by ESD Tools to create the inbound connection configuration, which contains the properties of connecting to EIS for the runtime requirement stored in the *.export file.

RMInboundConnectionConfiguration

This class represents the inbound connection configurations to RedMaintenance server. EMD Tools call the method createUnifiedProperties to get the connection

properties and show them on the editable panel. After users input values they want, EMD Tools stored those properties in the *.export file:

- ▶ `public PropertyGroup createUnifiedProperties()`
This method is for creating inbound property group for adapter runtime. Properties like connect string, user name and password need to be put in the property group for runtime use. Later, it is stored in the *.export file.
- ▶ `public PropertyGroup createActivationSpecProperties ()`
This method is for creating the properties that related to ActivationSpec class.
- ▶ `public PropertyGroup createResourceAdapterProperties()`
This method is for creating the properties that related to the resource adapter class.

RMIInboundServiceDescription

This class stores the methods that used to generate the service interface and business objects for inbound:

```
public void setFunctionDescriptions(MetadataSelection selection)
```

This method is called by `RMMetadataDiscovery.createServiceDescription()`. In this `setFunctionDescriptions` method, we get `MetadataSelection` as input, which contains all the metadata objects selected by users. We need to set most properties here, like BO location, relative path, BO attributes, namespace and so on for each metadata object.

RMMetadataTree

This class provides a tree structure for all the metadata objects. The EMD Tool displays this tree to the end user. Then, users can navigate this tree and select the metadata objects they want to import:

- ▶ `public RMMetadataTree(MetadataConnection connection)`
This construct method is called by `RMMetadataDiscovery` class to create `MetadataTree`. Here, we need to get information like EIS physic connection, BO prefix and so on from the input `MetadataConnection`.
- ▶ `public PropertyGroup createFilterProperties()`
This method is to create the filter properties for the EMD retrieving. For `RedMaintenance`, when user runs the query, EMD just returns those business objects whose name begins with the letters that specified in filter field.
- ▶ `public MetadataObjectResponse listMetadataObjects(PropertyGroup filterPG)`
This method is called when the user clicks the run query button. It lists all the business objects that match the filter as a tree structure. Also, it sets some

properties for each business object. After calling it, the business objects which are represented as `MetadataObject` are ready for the user to select as import. The sample is shown in Example 13-6.

Example 13-6 Sample `listMetaObjects` method implementation

```
{
    WBIMetadataObjectResponseImpl response = new
WBIMetadataObjectResponseImpl();
    ArrayList objects = new ArrayList();
    String filter = "";

    WBIMetadataDiscoveryImpl.getProgressMonitor().setMaximum(100);
    WBIMetadataDiscoveryImpl.getProgressMonitor().setMinimum(0);

    // Get Filter
    if (filterPG != null) {
        WBISingleValuedPropertyImpl filterProp = (WBISingleValuedPropertyImpl)
filterPG.getProperty("Filter");//$NON-NLS-1$
        if (filterProp != null) {
            filter = filterProp.getValueAsString();
        }
    }

    // Create sub-tree for data structure in RM server
    WBIMetadataDiscoveryImpl.getProgressMonitor().setNote("Retrieving data
structure in RM server...");
    WBIMetadataDiscoveryImpl.getProgressMonitor().setProgress(33);
    RMMetadataObject metadataObjectF = new RMMetadataObject();
    metadataObjectF.setLocation("Meta Data in RM Server");
    metadataObjectF.setDisplayName("Meta Data in RM Server");
    metadataObjectF.setParent(null);
    metadataObjectF.setHasChildren(true);
    metadataObjectF.setSelectableForImport(false);
    metadataObjectF.setType(MetadataObject.MetadataObjectType.FOLDER);
    metadataObjectF.setPrefix(prefix);
    addToTree(metadataObjectF.getLocation(),metadataObjectF);

    try {
        RMEMDInterface dataInterface=
appI.retrieveEMD(RMServerRMIInterface.RETRIEVE_FROM_RM_SERVER);
        Vector container = dataInterface.getStructures();
        ArrayList fixedChildren = new ArrayList();

        for(int i=0; i<container.size(); i++){
            RMEMDDefinition data = (RMEMDDefinition) (container.elementAt(i));

            if(filter!=null && filter.length()!=0 &&
!data.getComponentName().startsWith(filter)) continue;
        }
    }
}
```

```

        RMMetadataObject childObj = new RMMetadataObject();
        childObj.setLocation("MetaData in RM Server" +
data.getComponentName());
        childObj.setDisplayName(data.getComponentName());
        childObj.setParent(null);
        childObj.setHasChildren(false);
        childObj.setSelectableForImport(true);
        childObj.setType(MetadataObject.MetadataObjectType.OBJECT);
        childObj.setMetadata(data);
        childObj.setPrefix(prefix);
        addToTree(childObj.getLocation(), childObj);
        fixedChildren.add(childObj);
    }
    metadataObjectF.setChildMetadataObjectsList(fixedChildren);

} catch (Exception e) {
    e.printStackTrace();
}

// Create sub-tree for data structure in database
WBIMetadataDiscoveryImpl.getProgressMonitor().setNote("Retrieving data
structures in database...");
WBIMetadataDiscoveryImpl.getProgressMonitor().setProgress(66);
RMMetadataObject metadataObjectD = new RMMetadataObject();
metadataObjectD.setLocation("MetaData in Database");
metadataObjectD.setDisplayName("MetaData in Database");
metadataObjectD.setParent(null);
metadataObjectD.setHasChildren(true);
metadataObjectD.setSelectableForImport(false);
metadataObjectD.setType(MetadataObject.MetadataObjectType.FOLDER);
metadataObjectD.setPrefix(prefix);
addToTree(metadataObjectD.getLocation(), metadataObjectD);

try {
    RMEMDInterface dataInterface=
appI.retrieveEMD(RMServerRMIInterface.RETRIEVE_FROM_DATABASE);
    Vector container = dataInterface.getStructures();
    ArrayList dbChildren = new ArrayList();

    for(int i=0; i<container.size(); i++)
    {
        RMEMDDefinition data = (RMEMDDefinition) (container.elementAt(i));

        if(filter!=null && filter.length()!=0 &&
!data.getComponentName().startsWith(filter)) continue;

        RMMetadataObject childObj = new RMMetadataObject();

```

```

        childObj.setLocation("MetaData in Database" +
data.getComponentName());
        childObj.setDisplayName(data.getComponentName());
        childObj.setParent(null);
        childObj.setHasChildren(false);
        childObj.setSelectableForImport(true);
        childObj.setType(MetadataObject.MetadataObjectType.OBJECT);
        childObj.setMetadata(data);
        childObj.setPrefix(prefix);
        addToTree(childObj.getLocation(), childObj);
        dbChildren.add(childObj);
    }
    metadataObjectD.setChildMetadataObjectsList(dbChildren);

} catch (Exception e) {
    e.printStackTrace();
}

objects.add(metadataObjectF);
objects.add(metadataObjectD);
response.setObjects(objects);
WBIMetadataDiscoveryImpl.getProgressMonitor().setNote("Retrieving
Finished");
WBIMetadataDiscoveryImpl.getProgressMonitor().setProgress(100);
return response;
}

```

RMMetadataObject

This class is a representation of business objects in EIS. Usually one instance of MetadataObject represents one business object and its properties:

- ▶ `public MetadataImportConfiguration createImportConfiguration()`
This method creates a MetadataImportConfiguration, in which you can set the BO specific properties.
- ▶ `public MetadataObjectResponse getChildren(PropertyGroup arg0)`
This method returns the children MetadataObject list of the current MetadataObject. The input parameter is the filter properties.
- ▶ `public void populateBO(boolean topLevel)`
This method prepares the metadata for the current MetadataObject, such as prefix, attribute list, and so on.

The sample populateBO is shown in Example 13-7.

Example 13-7 Sample populateBO method implementation

```
{
    int j;
    ArrayList children = new ArrayList();

    if (prefix == null || prefix.trim().length() == 0) {
        prefix = "";
    }
    String busObjName = metadata.getComponentName();
    String boName = prefix + busObjName;

    this.setBOName(boName);

    LinkedHashMap boAttrs = new LinkedHashMap();
    Vector vector = metadata.getFieldDefinitions();

    for(int i=0; i<vector.size(); i++)
    {
        RMFieldDefintions field = (RMFieldDefintions)(vector.elementAt(i));
        RMASI asi = new RMASI();
        String attrName = field.getFieldName();
        String type = field.getFieldType();
        if(type!=null &&
type.equals(RMFieldDefinitionsConstants.TYPE_STRING))
            type = "string";
        asi.setAttributeName(attrName);
        asi.setType(type);
        asi.setKey(field.isKey());
        asi.setRequired(field.isRequired());

        boAttrs.put(attrName, asi);
    }

    //set the child container attribute
    if(metadata.getComponentName().equalsIgnoreCase("Apartment"))
    {
        RMASI asi = new RMASI();
        asi.setAttributeName("Maintenance");
        asi.setCardinality("N");
        for(j=0; j<sibling.size(); j++)
        {

            if(((RMMetadataObject)sibling.get(j)).getMetadata().getComponentName().equalsIg
noreCase("Maintenance"))
                break;
        }
    }
}
```

```

        if(j != sibling.size()) {
            ((RMMetadataObject)sibling.get(j)).setParent(this);
            ((RMMetadataObject)sibling.get(j)).populateBO(false);
            children.add((RMMetadataObject)sibling.get(j));

            String containerName =
                ((RMMetadataObject)sibling.get(j)).getMetadata().getComponentName();
            containerName = prefix + containerName;
            asi.setType(containerName.toLowerCase()+"."+containerName);
        }
        boAttrs.put("Maintenance", asi);
    }
    if(metadata.getComponentName().equalsIgnoreCase("Maintenance"))
    {
        RMASI asi = new RMASI();
        asi.setAttributeName("PartOrder");
        asi.setCardinality("N");
        for(j=0; j<sibling.size(); j++)
        {
            if(((RMMetadataObject)sibling.get(j)).getMetadata().getComponentName().equalsIgnoreCase("PartOrder"))
                break;
        }
        if(j != sibling.size()) {
            ((RMMetadataObject)sibling.get(j)).setParent(this);
            ((RMMetadataObject)sibling.get(j)).populateBO(false);
            children.add((RMMetadataObject)sibling.get(j));

            String containerName =
                ((RMMetadataObject)sibling.get(j)).getMetadata().getComponentName();
            containerName = prefix + containerName;
            asi.setType(containerName.toLowerCase()+"."+containerName);
        }
        boAttrs.put("PartOrder", asi);
    }

    this.setAttributes(boAttrs);

    if(children.size() > 0) {
        this.setHasChildren(true);
        this.setChildMetadataObjectsList(children);
    } else
        this.setHasChildren(false);

    if(topLevel)
        this.setParent(null);
}

```

RMMetadataObjectImportConfiguration

This class is for the BO specific properties. In RedMaintenance adapter EMD, we do not need such property. So, just extend WBIMetadataImportConfigurationImpl class.

RMMetadataSelection

This class contains all the Metadata Objects that user selected for import. Most business logic is implemented by adapter foundation classes WBIMetadataSelectionImpl. So we just simply create the following two methods:

- ▶ `public static String getNamespace()`
This returns the namespace for the selection.
- ▶ `public PropertyGroup createSelectionProperties()`
This method creates the properties for the whole selection, like service type, namespace, relative path and so on. ESD Tools calls this method to create property group, and then show them to the end user. After user inputs the values, ESD Tools sets them back to the property group.

RMASI

This class represents all the application-specific information that is set in the BO schema file. For RedMaintenance, ASI includes primary key, isRequired, Cardinality and so on.

RMDDataDescription

This class is used for getting the ASI metadata for business object and its attributes:

- ▶ `public Iterator getChildList()`
This method returns children list of the current BO.
- ▶ `public void prepareChildSchemaFiles()`
This method creates the data descriptions for all the children BOs of the current BO, and prepares schema files for them.
The sample is shown as Example 13-8.

Example 13-8 Sample prepareChildSchemaFiles method implementation

```
{
    for (Iterator i = this.getChildList(); i.hasNext();) {
        RMMetadataObject bo = (RMMetadataObject) i.next();
        RMDDataDescription dataDesc = new RMDDataDescription();
        dataDesc.setMetadataObject(bo);
        dataDesc.setRelativePath(getRelativePath());
        dataDesc.setName(getName().getNamespaceURI(), bo.getBOName());
    }
}
```

```

        if (MetadataObject.MetadataObjectType.OBJECT.equals(bo.getType()))
        {
            dataDesc.prepareChildSchemaFiles();
        }
        dataDesc.prepareSchemaFiles();
        //Get the schema definitions
        SchemaDefinition[] schemaFiles = dataDesc.getSchemaDefinitions();
        for (int j = 0; j < schemaFiles.length; j++) {
            SchemaDefinition definition = schemaFiles[j];
            put(definition.getNamespace(), definition.getLocation(),
definition.getContent());
        }
    }
}

```

- ▶ `public List getVerbs()`
This method returns the supported verb list for BO.
- ▶ `public List getNameSpaces()`
This method returns the namespace for ASI.
- ▶ `public List getImportNameSpaces()`
This method returns the namespace list that is imported into the current BO.
- ▶ `public WBIMetadata getMetadataForBusinessObject()`
This method returns the ASI metadata for business object.
- ▶ `public WBIMetadata getMetadataForAttribute(String attrName)`
This method returns the ASI metadata for attribute.

EMD.properties

This file contains all the constant strings that are used in EMD. Example 13-9 shows the sample.

Note: Some strings are mandatory for adapter foundation classes. So, if some strings cannot be found in this file, the runtime throws an exception. Example 13-9 is the sample properties file.

Example 13-9 Sample EMD.properties

```

# NLS_MESSAGEFORMAT_NONE
ConnectString = Connect String
ConnectStringDescription = The connect string that will be used to establish
the connection to RM server.
UserName = User Name

```

UserNameDescription = The user name that will be used to establish the connection to RM server.
Password = Password
PasswordDescription = The password that will be used to establish the connection to RM server.
Prefix = Prefix
PrefixDescription = Prefix that should be used for all Business Objects imported.
AdapterType = IBM RedMaintenance Adapter
AdapterTypeDescription = The resource adapter for RM Server.
ConnectionType = RM Connection
ConnectionTypeDescription = The connection to RM Server.
InboundProperties = Inbound Properties
InboundPropertiesDescription = The properties for the Inbound Connection.
ConnectionProperties = Connection Properties
ConnectionPropertiesDescription = The properties for the Metadata Connection.
ResourceAdapterProperties = Resource Adapter properties
ResourceAdapterPropertiesDescription = The properties for the Resource Adapter.
ManagedConnectionProperties = Managed Connection Factory Properties
ManagedConnectionPropertiesDescription = The properties for the Managed Connection Factory.

MetadataImportConfiguration = Metadata Import Configuration
MetadataImportConfigurationDescription = The properties for the Metadata Import Configuration.
SelectionProperties = Selection Properties
SelectionPropertiesDescription = The properties for the Metadata Selection.
ServiceType = ServiceType
ServiceTypeDescription = The service description that needs to be imported.
Namespace = Namespace
NamespaceDescription = The Namespace that should be used for the business objects.
Operations = Service Functions
OperationsDescription = The functions that should be created in the Service.
OutboundProperties = Outbound Properties
OutboundPropertiesDescription = The properties for Outbound Connection.
Miscellaneous = Miscellaneous
MiscellaneousDescription = Miscellaneous properties needed for the discovery service.
UserCredentials = User Credentials
UserCredentialsDescription = The credentials for user specific properties.
MachineCredentials = Machine Credentials
MachineCredentialsDescription = The credentials for machine specific properties.

LogMessages.properties

This file contains all the log messages that are used in the EMD. The log messages can be used by the LogUtil class.

13.2.7 Testing the EMD implementation by running it in WID

After implementing the above codes, we could launch the EMD to create service interfaces as well as BOs for RedMaintenance Adapter runtime (including outbound and inbound). Make sure that the RedMaintenance server and related database are running before you launch ESD Tools:

1. To Launch the ESD Tools, in the WebSphere Integration Developer, select **(Menu) → File → New → Project → Business Integration → Enterprise Service Discovery**. See Figure 13-6. If you cannot see the Business Integration item, ensure that the **Show All Wizards** option at the bottom of the panel is checked.

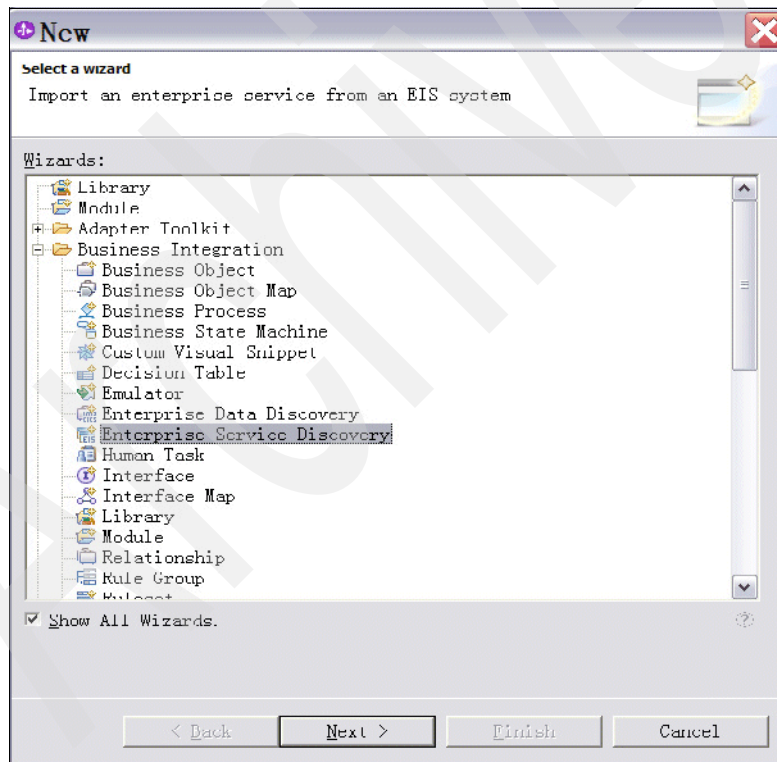


Figure 13-6 Create new Enterprise Service Discovery

2. Select **RedMaintenance EMD Adapter** and click **Next**. See Figure 13-7.

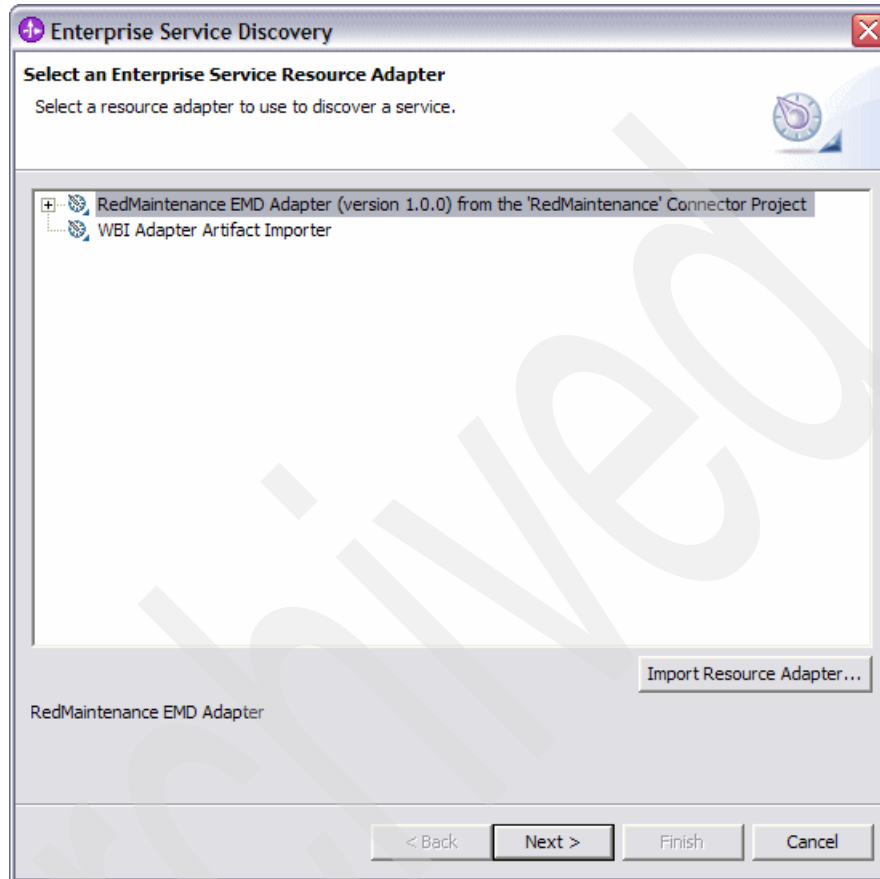


Figure 13-7 Select Enterprise Service Resource Adapter

3. In the configure settings dialog box, input the connection properties (including the RedMaintenance server URL and name) to your RedMaintenance server

as well as the prefix of BOs. You also can set logging properties here. Click **Next** after filling all the required properties. See Figure 13-8.

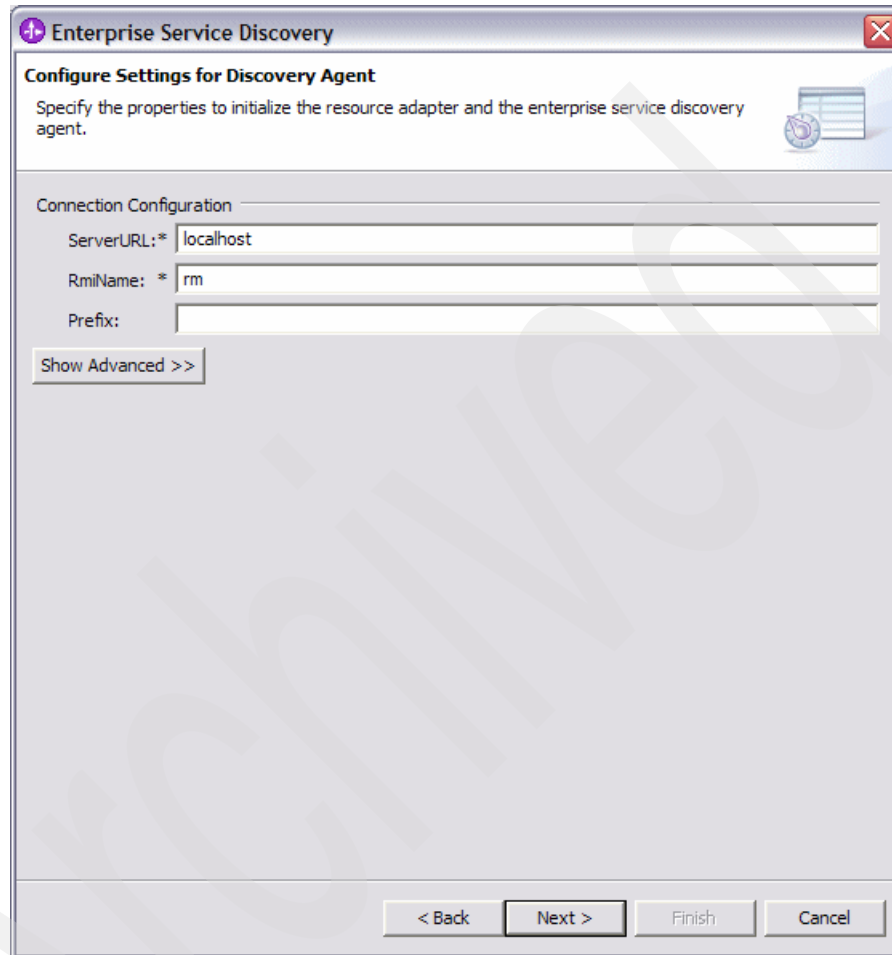


Figure 13-8 Configure settings dialog

4. In the Discovery dialog box, you can set Filter by clicking **Edit Query**. After that click **Run Query**. You can get the metadata objects listed in the window. See Figure 13-9 on page 423.

If you do not set the Filter, all available objects are displayed. If you set some letters in the Filter, only those objects with names beginning with the letters of the filter, are displayed.

Select the metadata object you want to import, and click **Add**. After adding all the objects you want, click the **Next** button. See Figure 13-9.

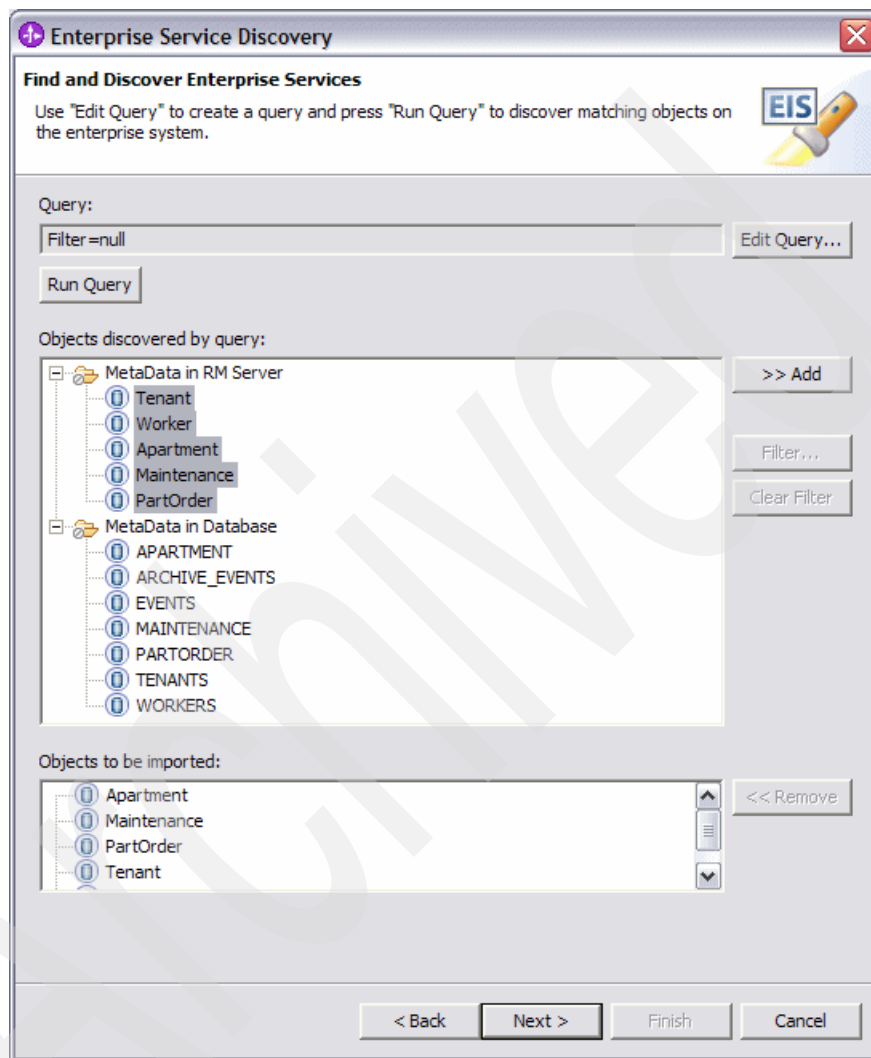


Figure 13-9 Discovery dialog box

Note: In Figure 13-9 there two folders in Object discovered by query:

- Metadata in RM Server
- Metadata in Database

Some EIS provide APIs that the adapter can use to query business objects in the EIS. If the EIS does not provide these APIs, the adapter might need to directly interact with EIS's database to query for business objects. RedMaintenance adapter provide example implementation for both scenarios. However, you should only pick the business objects generated from Metadata in RM Server for use with RedMaintenance inbound/outbound operations. This is because the business objects generated from RedMaintenance database needs further processing (more EMD code) before they are usable.

5. In the Configure Objects dialog box, first select service type as outbound or inbound. If you select **outbound**, the ESD creates *.import file. If you select **inbound**, the ESD create *.export file. Then, input the namespace and relative path for the BOs, as Figure 13-10 shows.

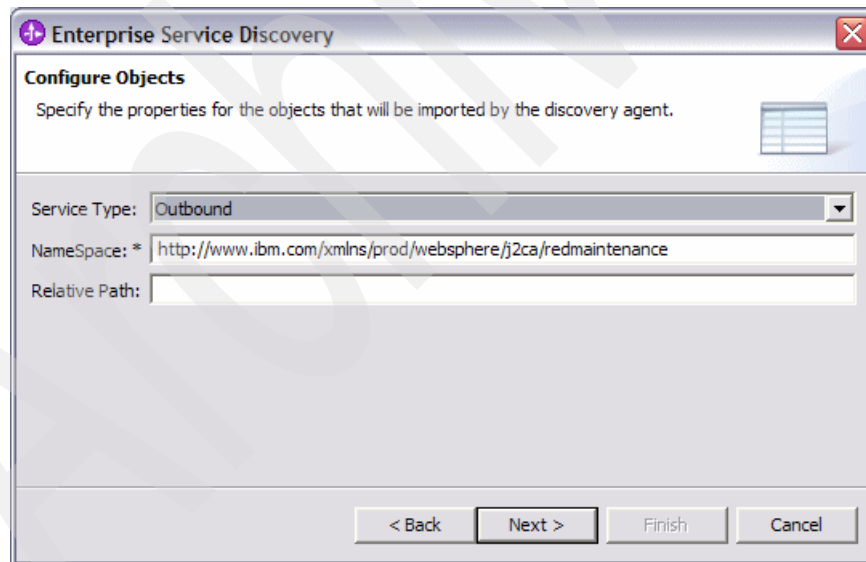
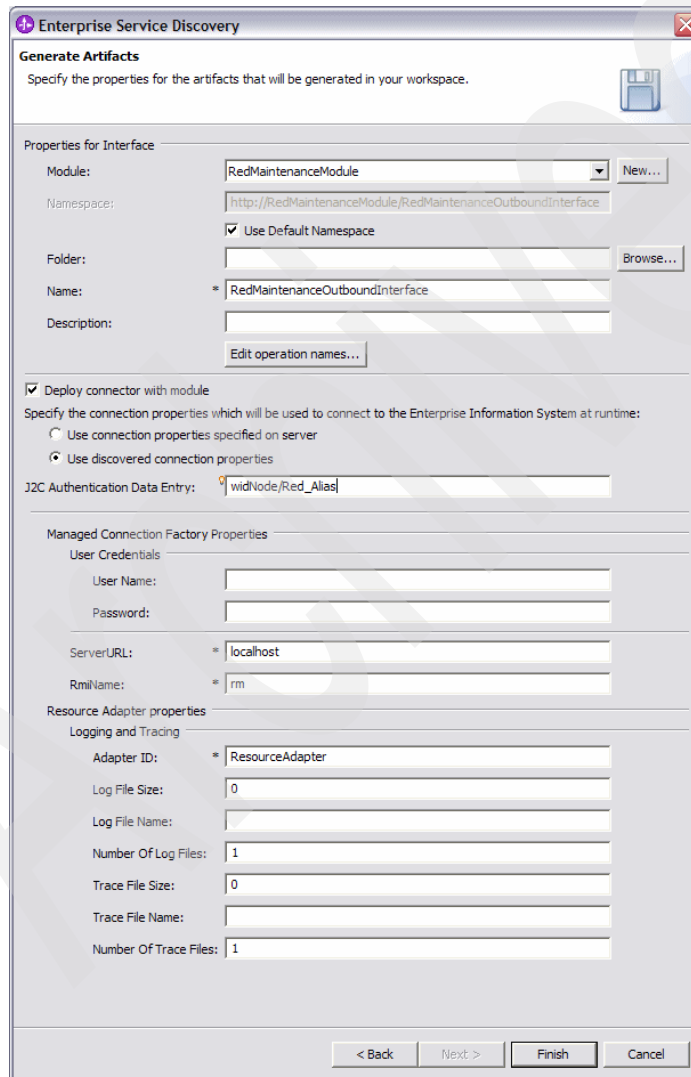


Figure 13-10 Configure Objects

6. In the generate artifacts dialog box shown in Figure 13-11, you can select the **business integration module**, in which the output files are put. Also, you can set the name of the import/export/wsdl files.

If in WebSphere Process Server, you have already created connection property group, you could set the JNDI name of that connection property group in JNDI Lookup Name field and select the **Use connection properties specified on server**. Otherwise, you can input values for those properties by selecting **Use discovered connection properties**.



The dialog box is titled "Enterprise Service Discovery" and has a sub-tab "Generate Artifacts". It contains several sections for configuring artifact generation and connection properties.

Properties for Interface

- Module: RedMaintenanceModule (dropdown menu)
- Namespace: http://RedMaintenanceModule/RedMaintenanceOutboundInterface (text field)
- Use Default Namespace: ☒ (checkbox)
- Folder: (empty text field)
- Name: * RedMaintenanceOutboundInterface (text field)
- Description: (empty text field)
- Edit operation names... (button)

☒ Deploy connector with module

Specify the connection properties which will be used to connect to the Enterprise Information System at runtime:

- ☐ Use connection properties specified on server
- ☒ Use discovered connection properties

J2C Authentication Data Entry: widNode/Red_Alias (text field)

Managed Connection Factory Properties

User Credentials

- User Name: (empty text field)
- Password: (empty text field)

ServerURL: * localhost (text field)

RmiName: * rm (text field)

Resource Adapter properties

Logging and Tracing

- Adapter ID: * ResourceAdapter (text field)
- Log File Size: 0 (text field)
- Log File Name: (empty text field)
- Number Of Log Files: 1 (text field)
- Trace File Size: 0 (text field)
- Trace File Name: (empty text field)
- Number Of Trace Files: 1 (text field)

Navigation buttons: < Back, Next >, Finish, Cancel

Figure 13-11 Generate artifacts dialog

After finishing all the fields, click **Finish**. All the files should be generated as shown in Figure 13-12. Here, we select the service type as **Outbound** and the data object **Tenant**.

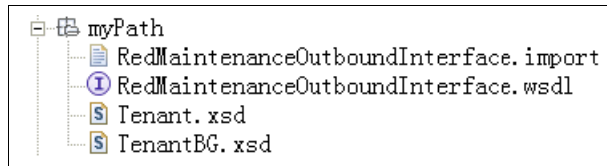


Figure 13-12 ESD generated service artifacts and business object definition

Here is a list of the sample files generated by RedMaintenance EMD:

► RedMaintenanceOutboundInterface.import

If we select the service type as **Outbound** during running EMD, the extension name of this file is .import. Otherwise, if we select **Inbound**, it is .export.

This file stores two groups of information:

- One is the connection configurations which is used by resource adapter to connect to EIS. It could be a property list or a simple JNDI name. If we use the JNDI name, we need to set corresponding connection configurations in the WebSphere Process Server Admin Console before deploying the application.
- The other is the method binding information. It indicates the method mapping information from the method name called by applications to the function name used by resource adapter.

This file is used by Service Component Architecture (SCA) components to connect to the resource adapter in order to interact with EIS. The SCA components can work as applications in the WebSphere Process Server.

In this document, we focus on the resource adapter development. You can refer to Chapter 1, “Introduction and adapter overview” on page 3 for basic knowledge about SCA and WebSphere Process Server. Refer to the SCA and WebSphere Process Server documents in the following links for further detail:

<http://www.redbooks.ibm.com/abstracts/redp4041.html?Open>

<http://www-128.ibm.com/developerworks/search/searchResults.jsp?searchType=1&searchSite=dW&searchScope=dW&query=sca&Search.x=0&Search.y=0&Search=Search>

Example 13-10 shows the .import file.

Example 13-10 RedMaintenanceOutboundInterface.import

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:import xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:eis="http://www.ibm.com/xmlns/prod/websphere/scdl/eis/6.0.0"
xmlns:ns1="http://RedMaintenanceModule/myPath/RedMaintenanceOutboundInterface"
```



```

        </methodBinding>
    </esbBinding>
</scdl:import>

```

► **RedMaintenanceOutboundInterface.wsdl**

This is the interface file that describes which methods can be called by SCA components to interact with EIS through the resource adapter. See Example 13-11.

Example 13-11 RedMaintenanceOutboundInterface.wsdl

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions
  xmlns:TenantBG="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/tenantbg"
  xmlns:intf="http://RedMaintenanceModule/myPath/RedMaintenanceOutboundInterface"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  name="RedMaintenanceOutboundInterface.wsdl"
  targetNamespace="http://RedMaintenanceModule/myPath/RedMaintenanceOutboundInterface">
  <types>
    <xsd:schema
      xmlns:tns="http://RedMaintenanceModule/myPath/RedMaintenanceOutboundInterface"
      xmlns:xsd1="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/tenantbg"
      elementFormDefault="qualified"
      targetNamespace="http://RedMaintenanceModule/myPath/RedMaintenanceOutboundInterface"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import
        namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/tenantbg"
        schemaLocation="TenantBG.xsd"/>
      <xsd:element name="createTenant">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="createTenantInput" type="xsd1:TenantBG"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="createTenantResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="createTenantOutput" type="xsd1:TenantBG"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="updateTenant">
        <xsd:complexType>
          <xsd:sequence>

```

```

        <xsd:element name="updateTenantInput" type="xsd1:TenantBG"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="updateTenantResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="updateTenantOutput" type="xsd1:TenantBG"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="retrieveTenant">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="retrieveTenantInput" type="xsd1:TenantBG"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="retrieveTenantResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="retrieveTenantOutput" type="xsd1:TenantBG"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="deleteTenant">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="deleteTenantInput" type="xsd1:TenantBG"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="deleteTenantResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="deleteTenantOutput" type="xsd1:TenantBG"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
</types>
<message name="createTenantRequest">
    <part element="intf:createTenant" name="createTenantRequestPart"/>
</message>
<message name="createTenantResponse">
    <part element="intf:createTenantResponse" name="createTenantResponsePart"/>
</message>
<message name="updateTenantRequest">
    <part element="intf:updateTenant" name="updateTenantRequestPart"/>

```

```

</message>
<message name="updateTenantResponse">
  <part element="intf:updateTenantResponse" name="updateTenantResponsePart"/>
</message>
<message name="retrieveTenantRequest">
  <part element="intf:retrieveTenant" name="retrieveTenantRequestPart"/>
</message>
<message name="retrieveTenantResponse">
  <part element="intf:retrieveTenantResponse"
name="retrieveTenantResponsePart"/>
</message>
<message name="deleteTenantRequest">
  <part element="intf:deleteTenant" name="deleteTenantRequestPart"/>
</message>
<message name="deleteTenantResponse">
  <part element="intf:deleteTenantResponse" name="deleteTenantResponsePart"/>
</message>
<portType name="RedMaintenanceOutboundInterface">
  <operation name="createTenant">
    <input message="intf:createTenantRequest" name="createTenantRequest"/>
    <output message="intf:createTenantResponse" name="createTenantResponse"/>
  </operation>
  <operation name="updateTenant">
    <input message="intf:updateTenantRequest" name="updateTenantRequest"/>
    <output message="intf:updateTenantResponse" name="updateTenantResponse"/>
  </operation>
  <operation name="retrieveTenant">
    <input message="intf:retrieveTenantRequest"
name="retrieveTenantRequest"/>
    <output message="intf:retrieveTenantResponse"
name="retrieveTenantResponse"/>
  </operation>
  <operation name="deleteTenant">
    <input message="intf:deleteTenantRequest" name="deleteTenantRequest"/>
    <output message="intf:deleteTenantResponse" name="deleteTenantResponse"/>
  </operation>
</portType>
</definitions>

```

► TenantBG.xsd

This is the business graph which extends the existing BusinessGraph.xsd. The BusinessGraph.xsd is shipped with WebSphere Integration Developer. In the business graph, besides the real business object link, usually there is a verb attribute. Refer to Chapter 3, “Service Data Objects” on page 53 for more

information about business objects and business graphs. See Example 13-12.

Example 13-12 TenantBG.xsd

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/tenantbg"
xmlns:tenant="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/tenant"
xmlns:bo="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0">
  <import
namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/tenant"
schemaLocation="Tenant.xsd"/>
  <import namespace="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0"
schemaLocation="../BusinessGraph.xsd"/>
  <complexType name="TenantBG">
    <complexContent>
      <extension base="bo:BusinessGraph">
        <sequence>
          <element name="verb" minOccurs="0" maxOccurs="1">
            <simpleType>
              <restriction base="string">
                <enumeration value="Create"/>
                <enumeration value="Update"/>
                <enumeration value="Retrieve"/>
                <enumeration value="Delete"/>
              </restriction>
            </simpleType>
          </element>
          <element name="Tenant" type="tenant:Tenant"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</schema>
```

► **Tenant.xsd**

This is the business object which contains the real data structure that is used for interacting between resource adapter and SCA components. Because it imports the ASI schema file, we could set any ASI into this business object. See Example 13-13.

Example 13-13 Tenant.xsd

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/tenant"
xmlns:tenant="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/tenant">
```

```

xmlns:tenant="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/tenant" xmlns:rmASI="asi">
<import
namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata"
schemaLocation=" ../RedMaintenanceASI.xsd"/>

<annotation>
<appinfo source="commonj.connector.asi">
<asi:annotationSet xmlns:asi="commonj.connector.asi"
asi:SURI="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata"
/>
</appinfo>
</annotation>
<complexType name="Tenant">
<annotation>
<appinfo
source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
<asi:BusinessObjectTypeMetadata
xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata"
">
<asi:ObjectName>Tenant</asi:ObjectName>
</asi:BusinessObjectTypeMetadata>
</appinfo>
</annotation>
<sequence minOccurs="1" maxOccurs="1">
<element name="Id" type="int" minOccurs="1" maxOccurs="1">
<annotation>
<appinfo
source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
<asi:PropertyTypeMetadata
xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata"
">
<asi:FieldName>Id</asi:FieldName>
<asi:PrimaryKey>true</asi:PrimaryKey>
</asi:PropertyTypeMetadata>
</appinfo>
</annotation>
</element>
<element name="Name" type="string" minOccurs="0" maxOccurs="1">
<annotation>
<appinfo
source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
<asi:PropertyTypeMetadata
xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata"
">
<asi:FieldName>Name</asi:FieldName>
</asi:PropertyTypeMetadata>
</appinfo>
</annotation>

```

```

</element>
<element name="ApartmentId" type="int" minOccurs="1" maxOccurs="1">
<annotation>
<appinfo
source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
<asi:PropertyTypeMetadata
xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
<asi:FieldName>ApartmentId</asi:FieldName>
</asi:PropertyTypeMetadata>
</appinfo>
</annotation>
</element>
<element name="EMail" type="string" minOccurs="0" maxOccurs="1">
<annotation>
<appinfo
source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
<asi:PropertyTypeMetadata
xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
<asi:FieldName>EMail</asi:FieldName>
</asi:PropertyTypeMetadata>
</appinfo>
</annotation>
</element>
<element name="Status" type="string" minOccurs="0" maxOccurs="1">
<annotation>
<appinfo
source="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata">
<asi:PropertyTypeMetadata
xmlns:asi="http://www.ibm.com/xmlns/prod/websphere/j2ca/redmaintenance/metadata
">
<asi:FieldName>Status</asi:FieldName>
</asi:PropertyTypeMetadata>
</appinfo>
</annotation>
</element>
</sequence>
</complexType>
</schema>

```

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material 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/SG246487>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG246387.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
SG246387.zip	All Zipped Code Samples

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Abbreviations and acronyms

ACID	Atonomy, Consistency, Isolation, Durability	ISV	Independent Software Vendors
API	an application programming interface	ITSO	International Technical Support Organization
ASI	application specific information	J2C	J2EE Connector Architecture
BG	business graph	J2EE	Java 2 Enterprise Edition
BPEL	Business Process Execution Language	J2EE	Java 2 Platform, Enterprise Edition
BPEL4WS	Business Process Execution Language for Web Services	JAAS	Java Authentication and Authorization Service
CBE	Common Business Event	JCA	J2EE Connector Architecture
CCI	Common Client Interface	JDBC	Java Database Connectivity
CEI	Common Event Infrastructure	JMS	Java Message Service
CORBA	Common Object Request Broker Architecture	JNDI	Java Naming and Directory Interface
CRUD	Create, Retrieve, Update, Delete	JSPs	Java Server Pages
DAS	data access service	JSR	Java Specification Request
EAI	Enterprise Application Integration	JSR47	Java Logging API
EAR	Enterprise Application Archive	ODA	Object Discovery Agent
EIS	Enterprise Information System	POJOs	Plain Old Java Objects
EJB	Enterprise JavaBeans	QoS	Quality of Service
EMD	Enterprise Metadata Discovery	RAD	Rational Application Developer
ERP	enterprise resource planning	RMI	Remote Method Invocation
ESD	Enterprise Service Discovery	RPC	Remote Procedure Call
HTTP	HyperText Transfer Protocol	SCA	Service Component Architecture
HTTPS	HyperText Transfer Protocol Secure	SCDL	Service Component Definition Language
IBM	International Business Machines Corporation	SCM	supply chain management
		SDO	Service Data Object
		SOA	IBM Service Oriented Architecture
		SOA	service-oriented architecture

SOAP	Simple Object Access Protocol
SPI	Service Provider Interface
SQL	Structured Query Language
UDB	Universal Database
W3C	World Wide Web Consortium
WID	WebSphere Integration Developer
WPS	WebSphere Process Server
WSDL	Web Services Description Language
WS-I	Web Services Interoperability
XSD	XML Schema Definition

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 on ordering these publications, see “How to get IBM Redbooks” on page 439. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Technical Overview of WebSphere Process Server and WebSphere Integration Developer*, REDP-4041

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ White paper, *Enterprise Metadata Discovery (EMD) Specification*
<ftp://www6.software.ibm.com/software/developer/library/j-emd/EnterpriseMetadataDiscoverySpecification.pdf>
- ▶ WebSphere Business Integration Adapters: An Adapter Development and WebSphere Business Integration Solution
<http://www.redbooks.ibm.com/redbooks/pdfs/sg246345.pdf>
- ▶ IBM WebSphere Adapter Toolkit
<http://www-128.ibm.com/developerworks/websphere/downloads/wat/>
- ▶ J2EE Connector Architecture
<http://java.sun.com/j2ee/connector/>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



WebSphere Adapter Development

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages

Index

A

- A process component 27
- A software adapter 35
- a wrapper 35
- Accessing the event store 234
- ActivationSpec 113
- ActivationSpec subclass 134
- adapter
 - building 244
 - design 237
 - development theory 1
 - Process Integration
 - business objects 31
 - service interface 77
 - specification 237
- adapter clients 75, 126
- adapter configuration properties 82
- adapter connection factory lookup 83
- adapter deployment descriptor 350
- adapter deployment descriptor schema 350
- adapter design technical assessment 238
- adapter developer 62, 118
- adapter foundation class 109, 149
- Adapter Foundation Classes 117
- adapter logging and tracing 147
- add dependent jar files 261, 355
- Address business object 67
- adoption of a standard specification 9
- adoption of SOA 13
- After-Image Create operation 86
- after-image Delete 90
- After-Image Update operation 88
- After-Image UpdateWithDelete operation 89
- after-images 63
- analyze EIS metadata 394
- apartment business object graphs 314
- application component 140
 - business objects 74
 - custom code 74
 - EIS resources 38
 - service interface 74
- application environment 225
- application programming interface (API) 238

- application server 140
 - application components 43
 - connection manager 41
 - message endpoints 44
 - orderly shutdown 39
 - resource adapter plugs 36
 - system contracts 36
- application sign-on 82, 246
- application-specific information
 - business objects 59
 - schema 401
- application-specific information (ASI) 53, 69, 138, 279, 298, 377
- ApplyChanges operation 86
- architecture 139
- Atomicity, Consistency, Isolation, Durability (ACID) 102
- attribute
 - application-specific information 70
- automation 6
 - self-configuring 8
 - self-healing 8
 - self-optimization 8
 - self-protecting 8

B

- basic concepts of WebSphere Adapter technology 4
- Bidi Support 256, 345
- bidirectional data communication 224
- blank, ignore, null properties 65
- Boolean flag 91
- build of custom WebSphere Adapters 4
- business attributes
 - focused 5
 - resilient 5
 - responsive 5
 - variable 5
- business drivers 238
- business drivers for On Demand Business 6
- business drivers of e-business on demand 6
- business graph
 - change summary portion 91

- data portion 91
- top-level verb 86
- business graph (BG) 59, 196
- business object
 - application-specific information 69
 - blank properties 65
 - ignore properties 65
 - naming 64
 - null properties 65
 - relationships 66
 - structure 65
- Business object (BO) 59
- business object architecture 57
- Business Process Execution Language (BPEL) 22
- Business Process Execution Language for Web Services (BPEL4WS) 22

C

- Command Pattern 95
- command pattern 99, 247, 252
 - brief overview 95
 - command class 99
 - Command Interpreter 96
 - Command Manager 96
 - processing delta object 98
 - processing snapshot object 96
 - sub-commands 96
- Common Business Event (CBE) 19, 154
- common client interface 38
- Common Client Interface (CCI) 44
- Common Event Infrastructure (CEI) 19, 154
- Common Object Request Broker Architecture (CORBA) 15
- common programming patterns 56
- Component Based Development 9
- component-managed sign-on mode 83
- concept of business object 26
- configuration properties 82
- connection instance 85
- connection management 83, 246
- connection management contract 40
- connection properties 253
- Create, Retrieve, Update, and Delete (CRUD) 64, 86, 239
- creating/constructing a new exception 149
- custom
 - adapter development 157
 - event class 132

- EventStore class 132
- operations 94
- resource adapter
 - creating 393
- Customer business object 67

D

- data 53
 - business objects 54
 - SDO 54
- data access code 55
- data access service (DAS) 56
- data concurrency control 56
- data-oriented Java APIs 55
- debugging 392
 - logging 336
 - tracing 336
- deleting events 360
- deliver the inbound event 126
- deltas 63
- deployment descriptor for EMD
 - creating 400
- develop an adapter 4
- development environment 159
- discovery-service.xml file 313
- dynamic and static data APIs 55

E

- EAI Tooling 141
- EIS
 - implement connection 262
 - inbound processing 355
- EIS assets 4
- EIS event
 - store 124
 - table 341
- EIS export binding 128
- EIS import
 - binding 80
- EIS import binding 80
- EIS resource 83
 - data exchange 47
- EIS service
 - connection information 80
 - export 128
 - import 80
- EMD - discovery 141
- EMD - Metadata edit 141

- EMD - runtime 141
- EMD - Tooling 141
- EMD implementation 420
- EMD logging 150
- EMD stub classes
 - implement 404
 - using WAT 394
- EMD tooling
 - utility APIs provided 403
- EMD.properties 418
- endpoint proxy 114
- Enterprise Application Integration (EAI) 44, 48, 138
- Enterprise Application Integration technology 9
- Enterprise Information System
 - data synchronization 60
 - resource adapter 38
- Enterprise Information System (EIS) 4, 31, 74, 108, 221
- Enterprise JavaBeans (EJBs) 19
- Enterprise Meta Discovery (EMD) 126
- Enterprise Metadata Discovery
 - component 48
 - J2EE role 143
 - specification 48, 80
- Enterprise Metadata Discovery (EMD) 76, 137–138, 252
- enterprise service bus (ESB) 15
- Enterprise Service Discovery
 - implementing 393
- Enterprise Service Discovery (ESD) 377
- Enterprise Service Discovery support 248
- Enterprise Service Discovery wizard 127
- Entity Relationship model 225
- error handling 87, 89, 92
- errors and exceptions 147
- event
 - filtering 359
 - handling 343
 - status 124
- event detection mechanism 116
- Event distribution 119
- Event Distribution table (EDT) 118, 130
- event management 114, 122
 - class diagram 117
 - error handling 124
- event retrieval mechanism 116
- event staging table 110
- event store
 - original event 124
- event store interface 116
- event triggering mechanism 236
- EventManager class 118
- EventStore interface 120
- Examples of business drivers 238
- Exception creation 148
- exception generation 149
- Exception objects 148
- exception traces 148
- Extensible Markup Language (XML) 54
- Extensible Markup Language (XML) schema 126
- external transaction manager 103

F

- features of message inflow contract 111

G

- generate outbound stub classes 255
- GSSCredential 44

H

- HyperText Transfer Protocol (HTTP) 14
- HyperText Transfer Protocol Secure (HTTPS) 14

I

- IBM On Demand Business 4
- IBM strategic vision 3
- IBM WebSphere Adapter 51
 - portfolio 47
 - Toolkit 174
- IBM WebSphere Adapter Toolkit 51
- IBM WebSphere Adapters 47
- implement Enterprise Metadata Discovery 393
- implementation of Service Oriented Architectures 3
- Imports and exports 24
- Inbound considerations 236
- inbound events 347
- inbound processing 107, 129, 343
 - basic understanding 107
 - detail theory 107
- inbound processing functionality 129
- inbound properties 130, 348
- Inbound scenario 242
- inbound stub classes 344
- Inbound support 248
- incoming business object 86
 - explicit comparison 88

- key values 86
- Independent Software Vendors (ISVs) 16
- infrastructure services 16
- initialization process 119
- input business
 - object 86
- integration
 - applications 7
 - data 7
 - people 7
 - processes 7
 - systems 7
- Integration developer 143
- integration developer 30
- integration services 16
- interactive install 159

J

- J2CA components 140
- J2EE application 138
- J2EE Connector Architecture (JCA) 38, 246
- J2EE developer 31
- Java Authentication and Authorization Service (JAAS) 43
- JAVA Connector Architecture (JCA) 4
- Java Database Connectivity (JDBC) 54, 135, 238
- Java Message Service (JMS) 14, 50
- Java Naming and Directory Interface (JNDI) 41
- Java Platform, Enterprise Edition (J2EE) 4
- Java Server Pages (JSPs) 24
- Java Specification Request (JSR) 55
- JCA
 - container 83
 - resource adapters 35
 - specification 35
 - target server 396
 - version 396
- JCA compliant application servers 37
- JCA compliant resource adapters 35
- JCA message inflow contract 111
- JCA resource adapter 51
- JNDI service 84

K

- key business and technical attributes 5
- key technological attributes 6
- key technology and architectural concepts 3

L

- list of integration services 16
- local transaction 248
- log messages 149
- logging 149
- LogMessages.properties 420

M

- message endpoint 112
 - factory 112
- message files 152
- metadata-driven adapter 241
- more information about SOA 17
- multiple-cardinality relationship 66
- multiple-cardinality relationships 67

N

- new adapter project
 - starting 395

O

- Object Discovery Agent (ODA) 50, 248
- Object Oriented Analysis and Design 9
- ObjectConverter utility class 276
- ObjectNaming utility class 273
- omponent-managed mode 82
- On Demand Business 3
- Open standards 9
- open standards 4, 6
- operations 64
- oreign-key attributes 66
- Outbound data 76
- outbound interaction 84
- outbound log messages 295
- outbound operation 242
- outbound operation invocation 80
- outbound operations 253, 268
- outbound request processing 73–74
 - implementing 251
- Outbound scenario 242
- outbound support
 - create 246
 - delete 246
 - retrieve 246
 - update 246
- overview of WebSphere Process Server 18

P

PasswordCredential 44
phantom mode 96
Plain Old Java Objects (POJOs) 19
polling events 359
primary-key attributes 66
problem determination 147, 149
problem resolution 149
property-level change 89

Q

Quality of Service (QOS) 245

R

Rational Application Developer (RAD) 29, 31
Redbooks Web site 439
RedMaintenance adapter 219, 237
 application sign-on 246
 connection management 246
 design 241
 Enterprise Service Discovery support 241
 inbound processing 343
 inbound support 241
 outbound request processing 251
 outbound support 246
 runtime support 246
 specification 241
 transaction management 241
RedMaintenance API 230, 239
 transaction behavior 241
RedMaintenance application 224, 238
 installing 226
 setting 226
RedMaintenance ASI schema 311
RedMaintenance outbound interface 315
relational data source 55
relationships between components 142
Remote Method Invocation (RMI) 228, 239, 253, 348
Remote Method Invocation/Internet Inter-ORB Protocol (RMI/IIOP) 15
Remote Procedure Call (RPC) 14
resource adapter 113, 141, 263
 common infrastructure programming model 45
 import model 48
 life cycle 39
 multiple EIS 37
 native format 80
 resource adapter base 110
Resource Adapter Deployment Descriptor 257
Resource adapter developer 143
Resource adapter modules 47
Resource adapter packaging 47
resource adapters 35
retrieve operation 91
RetrieveAll operation 92
 error handling 94
retrieving business objects 363
retrieving events 362
RMActivationSpec class
 revision 376
RMASI 417
RMBaseCommand class 281
RMCommandFactoryImpl class 279
RMConnection class 263
RMConnectionFactory class 264
RMCreateCommand class 283
RMDataDescription 417
RMDeleteCommand class 290
RMEvent subclass 356
RMEventStore
 full code 365
RMEventStore class 358
RMInboundConnectionConfiguration 410
RMInboundConnectionType 410
RMInboundServiceDescription 411
RMIMetadataTree 411
RMInteraction class 270
RMInteractionSpec class 272
RMManagedConnection class 266
RMManagedConnectionFactory class 264
RMMetadataDiscovery 405
RMMetadataEdit 405
RMMetadataObject 414
RMMetadataObjectImportConfiguration 417
RMMetadataSelection 417
RMOutboundConnectionConfiguration 406
RMOutboundConnectionType 406
RMOutboundServiceDescription 408
RMResourceAdapter 293
RMResourceAdapter class
 change 377
RMRetrieveCommand class 287
RMUpdateCommand class 285
root data object 57
Runtime support 246

S

- sample adapter
 - importing 215
- sample business scenario 221
- sample integration scenario 223
- sample scenario 222
- SCA and non-SCA services 24
- SCA artifacts
 - creation 377
- SCA artifacts and business objects 138
- SCA components 26
- SCA EIS export file 379
- SCA EIS service import file 317
- SCATestComponent 381
- SDO architecture 57
- SDO data object 57
- SDO design points 54
- SDO programming model 56
- security management contract 43
- Service 9
- Service choreography 11
- service component 19
- Service Component Architecture (SCA) 19, 201, 241
 - inbound processing 126
- Service Component Definition Language (SCDL) 25
- Service Data Object (SDO) 19, 45, 48, 53, 76
- service granularity
 - business functions 10
 - business processes 10
 - business transactions 10
 - technical functions 10
- service implementation types 29
- service interface 141
- service module 25
- Service Oriented Architecture 3
- service oriented architecture (SOA) 223
- Service Provider Interface (SPI) 45
- service-oriented architecture (SOA) 9
- Simple API for XML (SAX) 54
- Simple Object Access Protocol (SOAP) 14
- single-cardinality relationship 66
- single-cardinality relationships 66
 - data without ownership 67
- SOA (service-oriented architecture) 9
- SOA Core layer 20
- software (J2EE) developer 31
- staging table 130

- standard communication protocols 9
- standard JCA application programming model 84
- standard outbound operations 86
- Structured Query Language (SQL) 240
- supply chain management (SCM) 36
- supporting services layer 20
- Synchronous inbound processing 108
- system contract 38
 - connection management contract 39
 - life cycle management contract 39
 - message inflow 38
 - security contract 38
 - transaction inflow contract 38
 - transaction management contract 38
 - work management contract 39

T

- technical driver 238
- technology attribute
 - integration 6
- temporary SCA artifact 297
- Test inbound operations 385
- Tools developer 143
- Tracing 151
- Transaction management 248
- transaction management contract 42
- transaction processing (TP) 36
- transaction support 102
- transactional consideration 364
- two-phase commit protocol 103
- types of business objects 60

U

- underlying EIS
 - functionality 135
- update MANIFEST.MF file 297
- updating events 361
- using an existing adapter project 399
- using open standards 4

V

- verb
 - application-specific information 69
- Verbs and operations 64
- virtual dynamic organizations 8
- Virtualization 8
- virtualization 6

W

- Web Services Business Process Execution Language (WS-BPEL) 27
- Web Services Description Language (WSDL) 14
- Web Services Interoperability (WS-I) 14
- Web site 435
- WebSphere Adapter
 - business object model 61
 - inbound event notification 50
 - outbound support 50
- WebSphere Adapter architecture 49
- WebSphere Adapter business
 - object model 53, 61
 - object model align 63
 - object model support 65
 - object specification 64
- WebSphere Business Integration Adapters 50
- WebSphere Foundation Classes 76
- WebSphere Integration Developer
 - Enterprise Service Discovery wizard 76
- WebSphere Integration Developer (WID) 4, 49
- WebSphere Integration Developer (WID) 30
- WebSphere Integration Developer V6.0
 - installing 160
- WebSphere Process Server 17
 - bi-directional data communication 224
 - business object 57, 181
 - business object framework 58
 - data communication 53
 - inbound operation 242
 - integrated application 54
 - message end points/components 381
 - predefined endpoints 241
 - service component architecture 126, 252
 - service components 108
- WebSphere Process Server 75, 116, 164
 - architectural model 19
 - programming model 18, 20
 - service component architecture 22
- WSDL type interfaces 23

X

- XA transaction 43, 102, 248
- XML data source 55
- XML Schema Definition (XSD) 138



Redbooks

WebSphere Adapter Development

JCA 1.5 compliant Resource Adapter Architecture

Resource Adapter development examples and samples

Resource Adapter for WebSphere Process Server

This IBM Redbook shows you how to develop a JCA resource adapter based on IBM WebSphere Adapter Architecture. The custom adapter we build in this book implements J2EE Connector architecture (JCA), version 1.5, supporting managed, bidirectional connectivity between EISs and J2EE components in WebSphere Process Server.

The book is divided in two parts:

- ▶ The components of a WebSphere Adapter
- ▶ How to implement this adapter in an integrated scenario

WebSphere Adapter Architecture supports Service-Oriented Architecture (SOA), providing you with the ability to develop and modify integration applications dynamically. With SOA, you can also integrate existing applications with newer applications so that they work together transparently.

WebSphere Process Server V6.0.1 is a comprehensive SOA integration platform based on WebSphere Application Server V6. You can use WebSphere Process Server to develop and execute standards-based, component-based business integration applications.

Finally, we show you how to expose this adapter as an SCA service and use it with WebSphere Process server V6.0.1.

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