

IBM WebSphere Business Integration Adapters



Implementing Adapters with WebSphere Application Server

IBM WebSphere Business Integration Adapters



Implementing Adapters with WebSphere Application Server

Note!

Before using this information and the product it supports, read the information in "Notices" on page 173.

30September2004

This edition of this document applies to IBM WebSphere Business Integration Adapter Framework 2.6, and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this document, email doc-comments@us.ibm.com. We look forward to hearing from you

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2003, 2004. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this document	vii
What this document includes	vii
What this document does not include	vii
Audience	vii
Related documents	vii
WebSphere Business Integration adapters publications	viii
WebSphere Application Server publications	viii
System Manager publications	viii
WebSphere MQ publications	viii
Typographic conventions	viii

Summary of Changes	xi
New with WebSphere Business Integration Adapter Framework v 2.6.0	xi
New with WebSphere Business Integration Adapter Framework v 2.4.0	xi
WebSphere Business Integration Adapters v. 2.3.1	xi

Part 1. Overview and concepts 1

Chapter 1. Overview of WebSphere Business Integration adapters	3
A note about documents you need	3
What is the WebSphere business integration system?	4
What is an integration broker?	4
What are WebSphere Business Integration adapters?	5
How the WebSphere business integration system works	5
Data flow in the business integration system	7
Summary of the business integration process	11

Chapter 2. Business objects	13
Roles of a business object.	13
Event	13
Request.	13
Response	13
Structure of a business object	14
Business object type	14
Business object verbs	14
Business object attribute values.	15
Types of business objects	16
Business object definitions and business objects	16
Components of a business object definition.	17
Verbs	18
A closer look at business objects	19
Attribute organization	19
Application-specific information	19
Ways to create or modify business object definitions	21
Creating business object definitions	21
Modifying business object definitions.	22

Chapter 3. Connectors	23
Connector startup	23

Event notification	23
Setting up the application's event-notification mechanism	24
Detecting an event	27
Processing an event.	27
Guaranteed event delivery	29
Request processing	29
Verb-based processing.	30
Business object construction and deconstruction	31
Application-specific information for verbs	33
Connector configuration	34
Connector development	34

Chapter 4. Data transport and the integration broker 37

The role of the integration broker	37
Asynchronous data transport	37
Synchronous data transport	37
Interfaces for message exchange	37
Message formats.	38
Message queues	39
For more information	40

Part 2. Deployment and administration 41

Chapter 5. Overview: Implementing WebSphere Application Server as an integration broker 43

Chapter 6. Installing WebSphere Business Integration adapters 45

Installing for Windows systems.	45
Software Requirements	45
Installing the JDK	46
Installing WebSphere MQ.	46
Installing WebSphere Business Integration adapters	46
Installing plug-ins for WebSphere Studio Application Developer Integration Edition	46
Installing for UNIX systems	47
Software Requirements	47
Installing the JDK	48
Installing WebSphere MQ.	48
Installing WebSphere Business Integration adapters	49

Chapter 7. Configuring the WebSphere business integration system 51

Overview of configuration tasks	51
Creating business object definitions	52
Adding business objects to ICLs	53

Configuring WebSphere MQ queues for WebSphere Application Server	53
Creating the WebSphere MQ queues	54
Defining the queue configuration	56
Enabling the application for use with the connector	56
Configuring the connector	57
Running Connector Configurator	58
Setting Standard and Connector-Specific properties	59
Designating supported business objects	59
Specifying the queues to be used by the connector	60
Setting the connection mode with the queue manager	60
Configuring logging and tracing options.	61
Configuring the connector startup files, shortcuts, and environment variables	63
Using Visual Test Connector to verify your interfaces	64

Chapter 8. Deploying to WebSphere

Application Server.	65
Saving configurations as ICLs	65
Creating user projects in System Manager	67
Deploying user projects	67
Creating WebSphere Application Server applications	68
Agent Delivery (Asynchronous Event Delivery)	69
Agent Request (Synchronous event delivery)	84
Create an EJB project	85
Create an MDB	86
Hub One Way	90
Creating the EJB.	91
Create an application client project for testing	94
Hub Request	98
Creating the EJB project	99
Requirements for initiating business objects in interaction patterns	101
Adding the business object initialization library	101
Reserved values in business object handling	102
Transactional support.	103

Chapter 9. Administering the business integration system 105

Starting a connector	105
Stopping a connector	106
Creating multiple connector instances	107
Create a new directory	107
Clearing messages from WebSphere MQ queues	108
Managing log and trace files	108
Archival logging of log and trace files	109
Managing other files	110
Using Adapter Monitor and Fault Queue Manager	110
Adapter Monitor perspective	110
Setting Adapter Monitor preferences.	111
Loading an adapter	111
Adapter Monitor displays	112
Changing the state of an adapter	113
Using the Fault Queue Manager display	114
Handling failed events	114
Using Log Viewer to view connector messages	115

Setting Log Viewer preferences	115
Changing how messages are viewed.	118
Controlling the Log Viewer display output	119
Filtering messages.	120

Appendix A. WebSphere MQ message formats 123

Appendix B. Standard configuration properties for connectors 129

New properties.	129
Standard connector properties overview	129
Starting Connector Configurator	130
Configuration property values overview	130
Standard properties quick-reference	131
Standard properties	137
AdapterHelpName	137
AdminInQueue.	137
AdminOutQueue	137
AgentConnections.	137
AgentTraceLevel	138
ApplicationName	138
BiDi.Application	138
BiDi.Broker	138
BiDi.Metadata	138
BiDi.Transformation	139
BrokerType	139
CharacterEncoding	139
CommonEventInfrastructure	139
CommonEventInfrastructureContextURL	139
ConcurrentEventTriggeredFlows	139
ContainerManagedEvents	140
ControllerEventSequencing	141
ControllerStoreAndForwardMode	141
ControllerTraceLevel	141
DeliveryQueue	141
DeliveryTransport	142
DuplicateEventElimination	143
EnableOidForFlowMonitoring	143
FaultQueue	143
jms.FactoryClassName	143
jms.ListenerConcurrency	143
jms.MessageBrokerName	144
jms.NumConcurrentRequests	144
jms.Password	144
jms.TransportOptimized	144
jms.UserName	145
JvmMaxHeapSize	145
JvmMaxNativeStackSize	145
JvmMinHeapSize	145
ListenerConcurrency	145
Locale	145
LogAtInterchangeEnd	146
MaxEventCapacity	146
MessageFileName	146
MonitorQueue	147
OADAutoRestartAgent	147
OADMaxNumRetry	147
OADRetryTimeInterval	147
PollEndTime.	148

PollFrequency	148
PollQuantity	148
PollStartTime	149
RepositoryDirectory	149
RequestQueue	149
ResponseQueue	149
RestartRetryCount	150
RestartRetryInterval	150
ResultsSetEnabled	150
ResultsSetSize	150
RHF2MessageDomain	150
SourceQueue	151
SynchronousRequestQueue	151
SynchronousRequestTimeout	151
SynchronousResponseQueue	151
TivoliMonitorTransactionPerformance	151
WireFormat	151
WsifSynchronousRequestTimeout	152
XMLNameSpaceFormat	152

Appendix C. Connector startup options	153
Windows	153
UNIX	154

Appendix D. Using the Connector Script Generator tool	157
--	------------

Appendix E. Using Visual Test Connector	159
--	------------

Recommended testing procedure	159
Starting Test Connector	160
Shutting down Test Connector	160
Creating and editing connector profiles	160
Saving the connector definition to a file	160
Creating a new profile	161
Editing a profile	161
Deleting a profile	161
Emulating a connector	162
Working with business objects	162
Working with request business objects	162
Setting values for business object attributes	164
Saving a business object	165
Loading a business object	166
Deleting a business object	166
Accepting a request business object	166
Working with response business objects	166
Comparing business object instances	168

Appendix F. Upgrading WebSphere Business Integration adapters	169
--	------------

Index	171
------------------------	------------

Notices	173
Programming interface information	174

About this document

The IBM[®] WebSphere[®] Business Integration Adapter portfolio supplies integration connectivity for leading e-business technologies, enterprise applications, legacy applications and mainframe systems. The product set includes tools and templates for customizing, creating, and managing components for business integration.

What this document includes

This document describes how to install, configure, deploy, and manage WebSphere Business Integration adapters using WebSphere Application Server as the integration broker.

Note: Illustrations in this manual are only example used to show structure and concepts. They do not necessarily document specific business integration scenarios.

What this document does not include

This document does not describe deployment metrics and capacity planning issues, such as server load balancing, number of adapter processing threads, maximum and minimum throughputs, and tolerance thresholds.

Such issues are unique to each customer deployment and must be measured within or close to the exact environment where the adapter is to be deployed. You should contact your IBM services representative to discuss the configuration of your deployment site, and for details on planning and evaluating these kinds of metrics, given your specific configuration.

Audience

This document is for customers and consultants who are implementing or administering WebSphere Business Integration adapters for use with WebSphere Application Server. It is assumed that the reader already knows how to configure and administer WebSphere Application Server Enterprise and WebSphere Studio Application Developer Integration Edition, and has a solid understanding of WebSphere MQ messaging and message flows.

Related documents

Documentation you might need to deploy and manage WebSphere Business Integration adapters spans the libraries of several different products:

- Each of the WebSphere Business Integration adapters
- WebSphere Application Server
- WebSphere Studio Application Developer Integration Edition
- WebSphere MQ

Information about related books and instructions for accessing them are provided below.

WebSphere Business Integration adapters publications

The complete set of documentation describes the features and components common to all WebSphere Business Integration adapters installations, and includes reference material on specific components.

You can install the documentation or read it directly online at the following IBM Web site: <http://www.ibm.com/websphere/integration/wbiadapters/infocenter>

The site contains simple instructions for downloading, installing, and viewing the documentation.

WebSphere Application Server publications

You can browse and download documents about WebSphere Application Server and WebSphere Studio Application Developer Integration Edition at IBM's Web sites at: <http://www.ibm.com/software/webservers/appserv/library> and at <http://www.ibm.com/software/awdtools/studiointegration/library>

System Manager publications

Many configuration and administrative activities for WebSphere Business Integration adapters can be performed using a graphical user interface called System Manager. For more information about system manager, refer to the following two guides:

- *IBM WebSphere Interchange Server System Administration Guide*
- *IBM WebSphere Interchange Server Implementation Guide*

These are available at the following web site:

<http://www.ibm.com/websphere/integration/wicserver/infocenter>

WebSphere MQ publications

WebSphere MQ publications provide information about WebSphere MQ message formats and protocols. You can browse and download these documents from IBM's Web site at: <http://www.ibm.com/software/integration/mqfamily>.

Typographic conventions

This document uses the following conventions:

<code>courier font</code>	Indicates a literal value, such as a command name, filename, information that you type, or information that the system prints on the screen.
bold	Indicates a new term the first time that it appears.
<i>italic, italic</i>	Indicates a variable name or a cross-reference.
<i>blue text</i>	Blue text, which is visible only when you view the manual online, indicates a cross-reference hyperlink. Click any blue text to jump to the object of the reference.
{ }	In a syntax line, curly braces surround a set of options from which you must choose one and only one.
[]	In a syntax line, square brackets surround an optional parameter.
...	In a syntax line, ellipses indicate a repetition of the previous parameter. For example, <code>option[,...]</code> means that you can enter multiple, comma-separated options.

< >	In a naming convention, angle brackets surround individual elements of a name to distinguish them from each other, as in <server_name><connector_name>tmp.log.
/, \	In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All product pathnames are relative to the directory where the product is installed on your system.
<i>ProductDir</i>	Represents the directory where the product is installed. .
%text% and \$text	Text within percent (%) signs indicates the value of the Windows text system variable or user variable. The equivalent notation in a UNIX environment is \$text, indicating the value of the text UNIX environment variable.
Windows:	Paragraphs beginning with any of these indicate notes listing operating system differences.
UNIX:	
AIX:	
Solaris:	

Summary of Changes

This chapter summarizes the major changes made to *Implementing Adapters with WebSphere Application Server* guide for each release.

New with WebSphere Business Integration Adapter Framework v 2.6.0

The new adapter framework includes support for the following:

- WebSphere Application Server, versions 5.0.2 and 5.1.
- WebSphere Studio Application Developer Integration Edition, versions 5 and 5.1.1

New with WebSphere Business Integration Adapter Framework v 2.4.0

- Installation information updated for current release, including support of WebSphere Application Server 5.0.2.
- Installation information added for UNIX platforms.
- Information about starting and stopping adapters has been updated to include mention using Adapter Monitor.
- An appendix, “Upgrading WebSphere Business Integration adapters,” has been added.
- “Using Visual Test Connector” appendix has been updated.
- Changes made to accommodate new installation and packaging of adapters.

WebSphere Business Integration Adapters v. 2.3.1

The *Implementing Adapters with WebSphere Application Server* guide was new for this release.

Part 1. Overview and concepts

Chapter 1. Overview of WebSphere Business Integration adapters

The WebSphere Business Integration adapters use modular components and application-independent business logic. The **business integration system** supported by WebSphere Business Integration adapters is distributed and flexible, with customization features that make it possible to meet site-specific and application-specific needs.

This implementation guide explains how to deploy and manage a WebSphere business integration system that utilizes WebSphere Business Integration adapters with WebSphere Application Server. The guide is divided into two parts:

- Part 1 presents a conceptual overview of the WebSphere Business Integration adapters portfolio, its major components, and the process by which adapters interact with an integration broker to create a business integration system. Information in Part 1 applies to WebSphere Business Integration adapters used with WebSphere Application Server as the integration broker and (except where noted as specific to WebSphere Application Server) with other integration brokers.
- Part 2 offers task-oriented information to help you install, configure, and administer WebSphere Business Integration adapters specifically with WebSphere Application Server as the integration broker. Part 2 is specific to the use of adapters with WebSphere Application Server and WebSphere Studio Application Developer, Integration Edition.

This chapter describes the architecture of a business integration system and introduces the components of a WebSphere Business Integration adapter. It contains the following sections:

- “What is the WebSphere business integration system?” on page 4
- “What is an integration broker?” on page 4
- “What are WebSphere Business Integration adapters?” on page 5
- “How the WebSphere business integration system works” on page 5

A note about documents you need

To perform all the tasks necessary to deploy a business integration adapter, you need to use this book together with other books in the WebSphere Business Integration adapters library, particularly:

- The *Business Object Development Guide*
- The adapter user guides for the adapters you are deploying.

If you are developing a custom adapter, you also need to use either or both of the following books:

- *Connector Development Guide for Java*
- *Connector Development Guide for C++* .

For a complete listing of the books in the library, see “WebSphere Business Integration adapters publications” on page viii.

What is the WebSphere business integration system?

To compete effectively in e-business, an enterprise must meet two challenges:

- It must move business information among diverse sources to perform business exchanges.
- It must process and route business information among disparate applications in the enterprise environment

A business integration system addresses both these needs with flexibility and extensibility.

At the highest-level, the **WebSphere business integration system** consists of an integration broker and a set of adapters that allow heterogeneous business applications to exchange data through the coordinated transfer of information, in the form of business objects.

Figure 1 shows a simplified representation of a WebSphere business integration system.

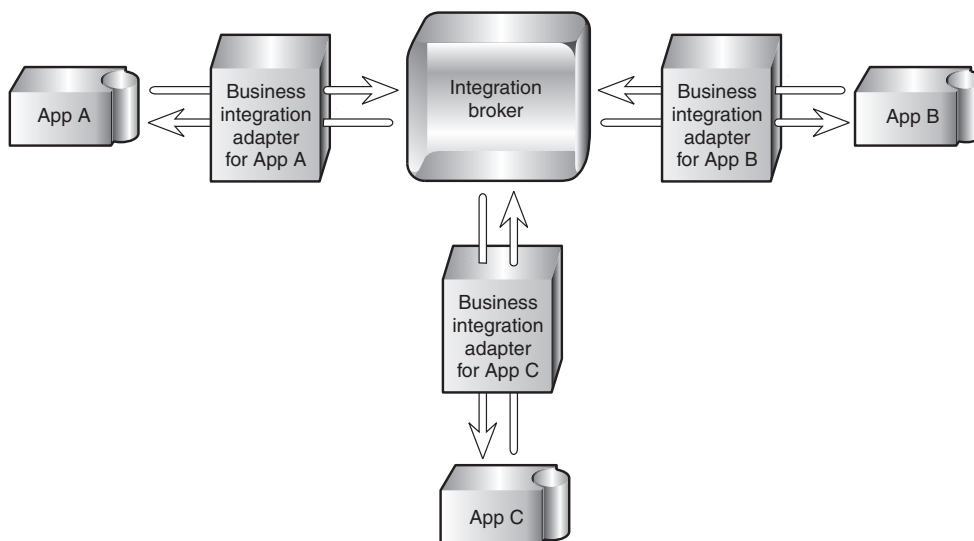


Figure 1. High-level view of a WebSphere business integration system

What is an integration broker?

An integration broker enables diverse applications to exchange information in dissimilar forms by handling the processing required for the information to arrive in the right place and in the correct format. In addition, a broker may facilitate the application of user-defined rules or business logic to the processing of the data. Data exchange is performed by the integration broker without requiring applications to have any knowledge of the data conventions or requirements of the applications receiving their data.

What are WebSphere Business Integration adapters?

The WebSphere Business Integration adapters consist of a collection of software programs, application programming interfaces (APIs), and tools you can use to enable applications to exchange business data through an integration broker. Each business application requires its own application-specific adapter to participate in the business integration system.

Each adapter in WebSphere Business Integration adapters portfolio includes:

- A connector that links the application to the integration broker
- Tools with graphical user interfaces to help you configure a connector and create the business object definitions needed for the application.
- An Object Discovery Agent (ODA), which runs against an application's data store to create business object definitions, which you can then refine. Note that WebSphere Business Integration adapters for some applications do not include an ODA.
- An Object Discovery Agent Development Kit (ODK), which consists of a set of APIs you can use to develop an ODA.

A separately-available Adapter Development Kit (ADK) provides a framework for developing custom adapters in cases where a prebuilt connector for a particular legacy or specialized application is not available from IBM.

How the WebSphere business integration system works

In the WebSphere business integration system implemented with WebSphere Application Server, connectivity for moving data between applications and WebSphere Application Server is supplied by **connectors** using WebSphere MQ as the Java Message Service (JMS) provider. A connector can reside on any machine from which it can access the necessary queues and communicate with the application.

Each connector consists of two parts—the **connector framework** and the **application-specific component**:

- The connector framework interacts with the integration broker using WebSphere MQ queues.
- The application-specific component interacts directly with an application.

The subcomponents of a connector are shown in Figure 2 on page 6.

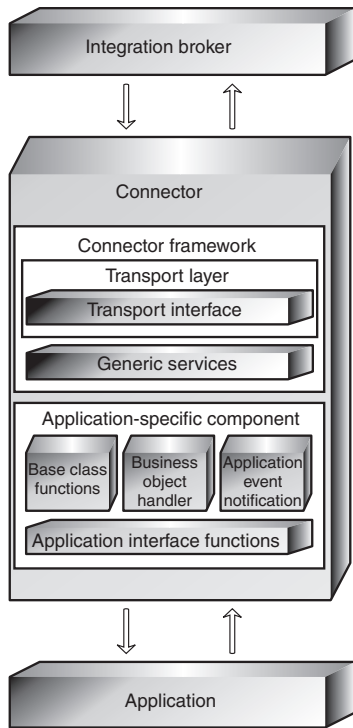


Figure 2. Subcomponents of a connector

Data is exchanged between applications by means of **application-specific business objects**, which are transported between the connector framework and the integration broker as WebSphere MQ JMS messages (also referred to as business object messages).

Business objects encapsulate and transmit business data for the several purposes. They convey:

- New or changed data from a source application to a destination application.
- Requests for data made by a source application to a destination application.
- Data returned by an application in response to a request for data.

Instructions, associated with each piece of data, encoded as **metadata**, specify the location in the application's database where the data is to be found, created, or updated. New instances of business objects are created by the application-specific component based on templates called **business object definitions**, which specify the structure and organization of the business object's attributes, values, and metadata.

Because application-specific information and other metadata in the business object definition guide the actions of the application-specific component, such an application-specific component's behavior can be described as **metadata-driven**. An application-specific component that is metadata-driven is flexible because it has no hard-coded instructions for each type of business object that it supports. Without recoding or recompiling, the application-specific component automatically supports new business object definitions, as long as the corresponding application data can be accurately described by the connector's metadata syntax.

Data flow in the business integration system

In the business integration system, data flow—the movement and processing of data sent from one application or entity to another— can occur either as an asynchronous or a synchronous exchange between applications, either on a local network or across the Internet.

An application might need to exchange data with another application to communicate changes in its data store or to obtain data.

The exchange of data in the business integration system consists of these steps:

1. Event notification
2. Integration broker processing
3. Request processing

Each of these is explained in more detail below.

Event notification

The process of conveying changed application data to the integration broker is called **event notification**. Most applications that participate in the business integration system are modified during the configuration process to include an **event store**, such as a table for logging the application's data changes and data requests. To detect that an application has newly changed data to share or that it needs information from another application, the connector framework initiates a **poll call** at periodic intervals. The poll call asks the application-specific component to check for changes to the application's event store.

If there has been a change since the last poll call, the application-specific component determines if a business object definition exists to represent the changed data or the data request. The presence of a suitable business object definition in the connector's local repository is an indicator that this particular change or request needs to be communicated to another application. The application-specific component sends the application data, in the form of a business object, to the connector framework. This is referred to as an **event delivery**, because a change to an application's data or a request for data is considered an event.

Figure 3 on page 8 shows a connector and its supporting infrastructure detecting a change to the application's data store and constructing an application-specific business object to convey the changed data to the integration broker.

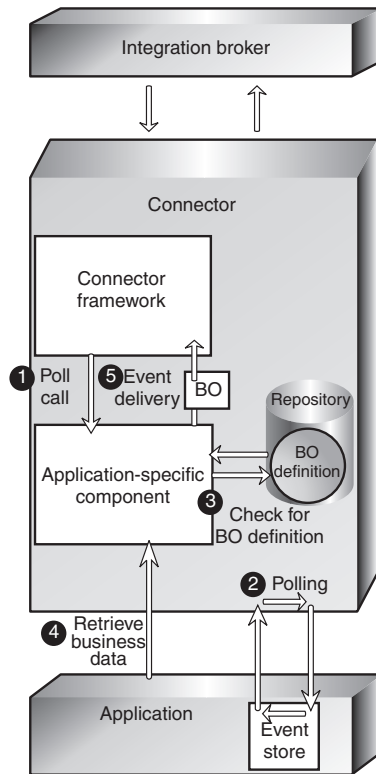


Figure 3. The connector detecting and delivering an event.

The numbers in the figure show the sequence of steps:

1. The connector framework initiates to the application-specific component to have it check for changes to the application's event store.
2. The application-specific component polls for changes to the application's event store.
3. The application-specific component determines whether the changed data maps to a supported business object definition.
4. The application-specific component instantiates a business object and uses it to retrieve the changed data.
5. The application-specific component initiates an event delivery to transfer the business object to the connector framework.

When the connector framework of the source application receives the application-specific business object, it converts the business object to a WebSphere MQ message that can be placed on a WebSphere MQ queue for receipt by the integration broker. A **data handler** is used by the connector framework to transform the business object into a message in the appropriate XML-based wire format for the destination WebSphere MQ queue. Figure 4 on page 9 shows this process.

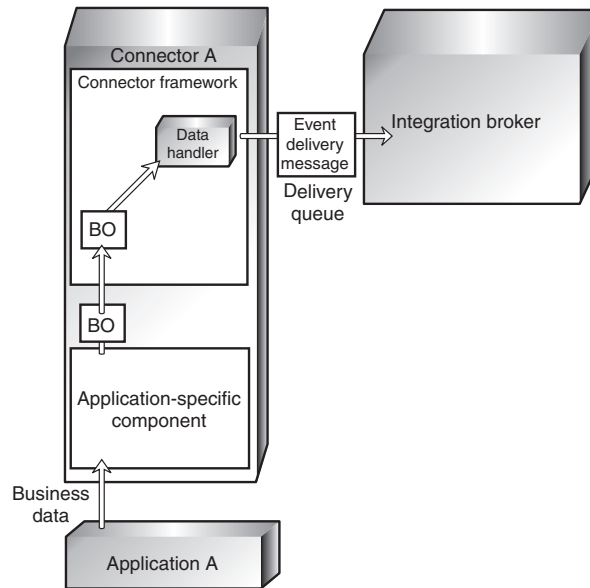


Figure 4. The connector framework transforming a business object into an MQ message.

Integration broker processing

After the message is placed on the WebSphere MQ event delivery queue for the integration broker, the integration broker removes the message for processing. In a WebSphere Application Server environment, this task, and the processing that follows, are performed by J2EE components that have been created using WebSphere Studio Application Developer, Integration Edition. Messages picked up from the queue are handled by MDBs and processed by EJBs.

The processing in the integration broker produces a message that is to be sent to the destination application. The message, called a **request** (or HubRequest in an WebSphere Application Server environment), is placed on a WebSphere MQ request queue to be transferred to the connector framework of the destination application.

Request processing

Once the request has been placed on the queue for the destination connector, a **listening mechanism** notifies the connector framework of the destination application's adapter that a WebSphere MQ message has arrived on its request queue and needs to be processed. The connector framework invokes the data handler to convert the WebSphere MQ message into a business object that can be processed by the destination application, as shown in Figure 5 on page 10.

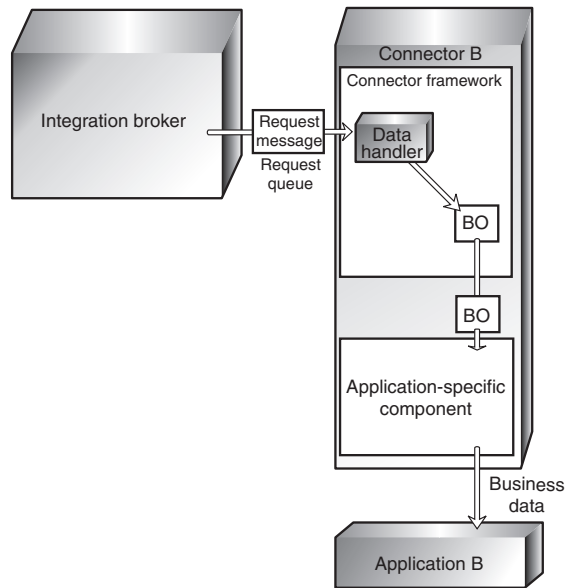


Figure 5. Request processing by the connector.

In some cases, the request might require a **response** from the destination application. Generally, a response is used to:

- Return data that the source application has requested from the destination application
- Return information to the source application about a new business entity (such as a customer or an order) that the source application has asked the destination application to create.

If a response is needed, the application-specific component modifies the request business object to carry the information and sends the business object back to the connector framework. The connector framework calls the data handler to convert the business object to a WebSphere MQ message and places the message on the reply-to queue specified in the originating request message. A correlation ID in the response message identifies the message to which it is responding. Figure 6 on page 11 illustrates how response processing is performed.

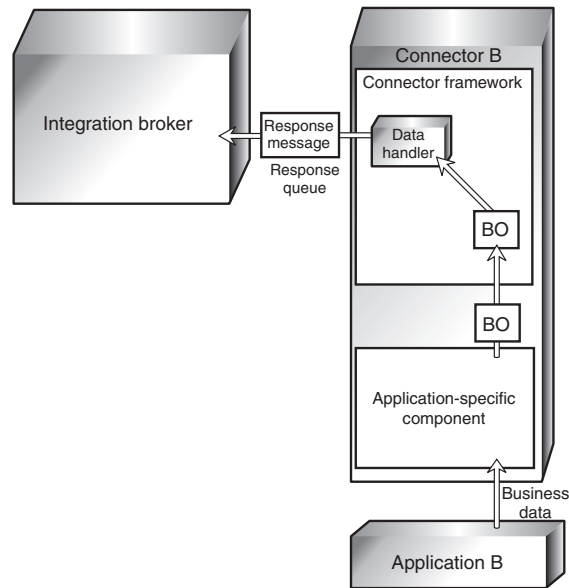


Figure 6. Response processing by the connector.

Summary of the business integration process

Now that you have learned about each step in the business integration process, you can step back for a look at the system as a whole. Two scenarios are presented below.

Example of sending changed data to another application

As an example, here are the steps by which the business integration system enables application A to send changed data to application B for synchronization:

1. Connector A's application-specific component detects a change to data in application A. It determines that a business object definition exists for communicating this change and uses the business object definition to construct a business object to carry the changed information.
2. The application-specific component passes the business object to the connector framework.
3. The connector framework invokes the data handler to transform the business object into a WebSphere MQ message of the correct XML-based wire format and places the message on a WebSphere MQ queue for the integration broker.
4. The integration broker receives and processes the message. In a WebSphere Application Server environment, this takes place when an MDB receives the message from the queue and routes it to the appropriate EJB for processing.
5. After processing the message that originated from the connector for application A, the integration broker places the resulting message on the WebSphere MQ queue for the connector for application B.
6. Connector B's connector framework removes the message from the queue and calls the data handler to convert it to a business object that can be processed by the application-specific component.
7. Application B updates its customer information to reflect the change of address.

If application A were requesting data from application B instead of notifying it of a data change, application B would need to send a response back to application A. The following example illustrates this scenario.

Example of obtaining data from another application

Here are the steps by which the business integration system enables application A to retrieve information about a customer's most recent purchase from application B.

1. Connector A's application-specific component detects that application A has requested data from application B. It determines that a business object definition exists for communicating this request and uses the business object definition to construct a business object for the requested information.
2. The application-specific component passes the business object to the connector framework.
3. The connector framework invokes the data handler to transform the business object into a WebSphere MQ message of the correct wire format and places the message on a WebSphere MQ queue for the integration broker.
4. In a WebSphere Application Server environment, an MDB in the integration broker receives the message and invokes an EJB that will process it.
5. One or more EJBs process the incoming message and then place the resulting outgoing message on the WebSphere MQ queue for the connector for application B.
6. Connector B's connector framework removes the message from the queue and calls the data handler to convert it to a business object that can be processed by the application-specific component.
7. Connector B's application-specific component retrieves the information specified in the request and passes it back to the connector framework as a business object.
8. Connector B's connector framework invokes the data handler to transform the business object to a response message and places it on the reply-to queue specified in the originating request.

The chapters that follow describe in more detail the business integration components introduced here and the process by which they enable applications to share data.

Chapter 2. Business objects

A business object reflects a data entity—a collection of data treated as a unit. For example, a data entity can be equivalent to an employee record, containing all the basic information about the employee - the name, address, telephone number, employee number, position code, salary, and so forth.

The business integration system creates business objects that reflect the information contained in entities. In this book, a data entity is often referred to in the context of the kind of business information it contains—for example, an *employee entity* or a *customer entity*.

Business object definitions are the templates from which the application-specific component creates a particular instance of a business object.

This chapter introduces business objects in more detail and explains how they are used by the business integration system to carry data between applications. It includes the following sections:

- “Roles of a business object”
- “Types of business objects” on page 16
- “Business object definitions and business objects” on page 16
- “A closer look at business objects” on page 19
- “Ways to create or modify business object definitions” on page 21

Roles of a business object

A business object can act as an event, a request, or a response.

Event

A business object can report the occurrence of an **application event**, an operation that affected a data entity in an application. The application event might be the creation, deletion, or change in value of that collection of data.

When a connector detects an application event and sends a business object to the integration broker, the role of the business object is to represent the event. So, it is called an **event** in the business integration system.

For example, a connector might poll an application for new employee entities on behalf of the integration broker. If the application creates a new employee entity, the connector sends an event business object to the integration broker.

Request

Requests are typically generated as follows. The integration broker sends a business object message as a **request** to the connector framework, instructing it have the application-specific component insert, change, delete, or retrieve some data in an application.

Response

When a connector finishes processing a request, it usually returns a **response** to the integration broker. For example, when a connector receives a request to create

an employee record in the destination application, it sends a business object with the created employee data and a status indicator that shows that the create was successful.

Structure of a business object

A business object is a self-describing unit that contains a type (its name), processing instructions (a verb), and data (attribute values).

Figure 7 is an example of a simple business object, showing its type, verb, and attribute values.

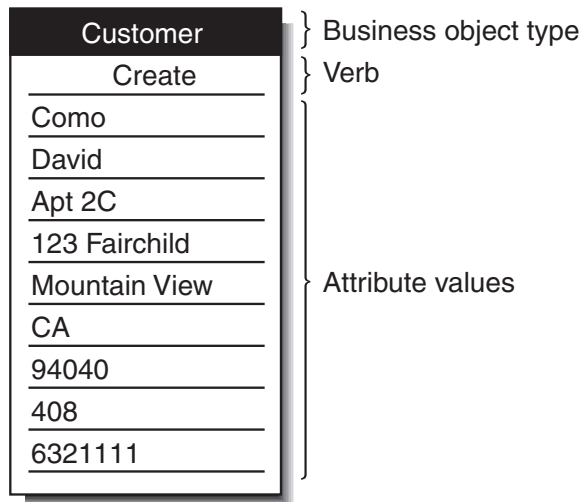


Figure 7. Business object components.

The next sections describe these components.

Business object type

Each business object has a type name that identifies it within the business integration system. For example, the type might be Customer, Employee, Item, or Contract.

Business object verbs

A business object verb specifies an action in relation to the attribute values. The verb can indicate various types of actions, depending on the role of the business object. Table 1 lists the three business object roles and describes the meaning of the verb in a business object that has each role.

Table 1. Meanings of business object verbs.

Role of business object	Meaning of verb
Event	Describes what happened in an application. For example, in an event, the Create verb indicates that the source application created a new data entity.
Request	Tells the connector how to interact with the application in order to process the business object. For example, the Update verb is a request to the connector to update the data entity.

Table 1. Meanings of business object verbs. (continued)

Role of business object	Meaning of verb
Response	Lists the verb specified in the associated request. For example, in a response, the Retrieve verb indicates that the connector obtained the attribute values from the application.

Note: The IBM convention is to use the format *business-object-type.verb* to indicate a particular type of business object with a particular verb. For example, Customer.Create is a Customer business object with the Create verb.

Business object attribute values

A business object contains **attribute values** that represent data fields associated with the data entity, such as Last Name, First Name, Employee ID, or Invoice Status.

Some attributes, instead of containing data, contain **child business objects** or **arrays of child business objects**. Figure 8 illustrates the structure of a Contract business object. The Line Item information in the contract is in an array of child business objects.

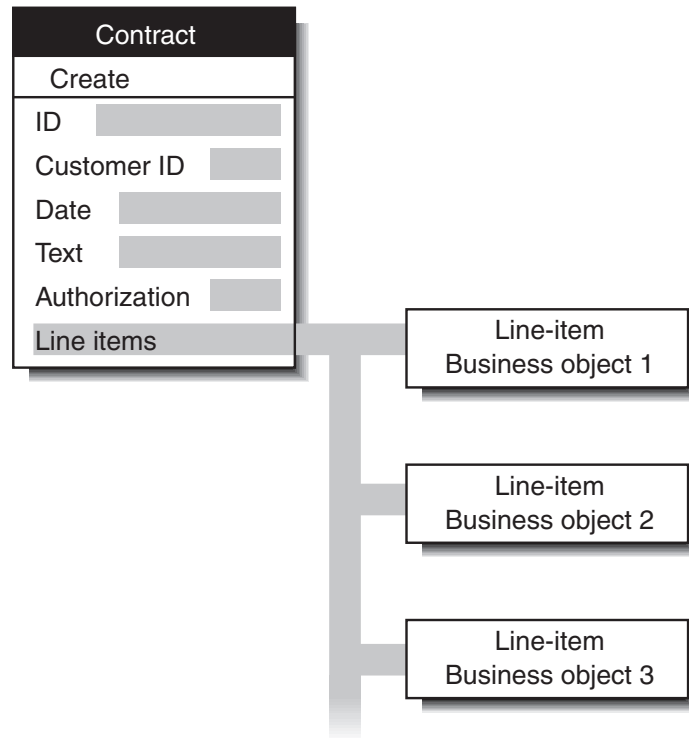


Figure 8. Business object with child business objects.

A business object that contains child business objects or arrays of child business objects is a **hierarchical business object**. One whose attributes contain only data is a **flat business object**.

Types of business objects

There are two types of business objects: application-specific and generic. Application-specific business objects are used in any WebSphere business integration system, regardless of which integration broker is being used. Generic business objects are used only if the integration broker is WebSphere InterChange Server (ICS). When WebSphere Application Server is the integration broker, only application-specific business objects are used. Therefore, all references to business objects throughout this book refer to application-specific business objects. Many of the books in the IBM WebSphere Business Integration (WBI) Server documentation set cover both environments and therefore refer to both types of business objects.

- **Application-specific business objects** reflect the data entity attributes and the data model of a specific application or other programmatic entity.
- **Generic business object** contain sets of business-related attributes that are common across a wide range of applications, and are not tied to any specific application's data model. Generic business objects are not used when WebSphere Application Server is the integration broker, but are discussed in the books that are included in both the WebSphere Business Integration adapters library and the WebSphere InterChange Server library.

When an application-specific component detects an application event such as an update, it retrieves the appropriate data entity from the application and transforms it into a business object.

Note: When documentation refers to a business object whose name includes an application name, such as `Clarify_Contact` or `Oracle_Customer`, it refers to an application-specific business object. A `Clarify_Contact` business object, for example, contains the set of information that the Clarify application stores about a contact. In another application, a contact entity might store a somewhat different set of information, store the information in a different order or format, or have a different name.

After an application-specific component has built a business object, it sends the business object to the connector framework. The connector framework calls the data handler to convert the business object to a WebSphere MQ message to be dispatched to the integration broker.

Business object definitions and business objects

Chapter 1, "Overview of WebSphere Business Integration adapters," on page 3, introduced business objects but only mentioned briefly the distinction between business object definitions and instances of the business objects themselves. Let's look more closely at that distinction now:

- A **business object definition** specifies the types and order of information in each entity, and the verbs that it supports. The local repository for the connector stores business object definitions.
- A **business object** is an instance of the definition, containing actual data. Business objects are created at runtime and not stored in the repository.

Figure 9 on page 17 illustrates the relationship between a business object definition and a business object.

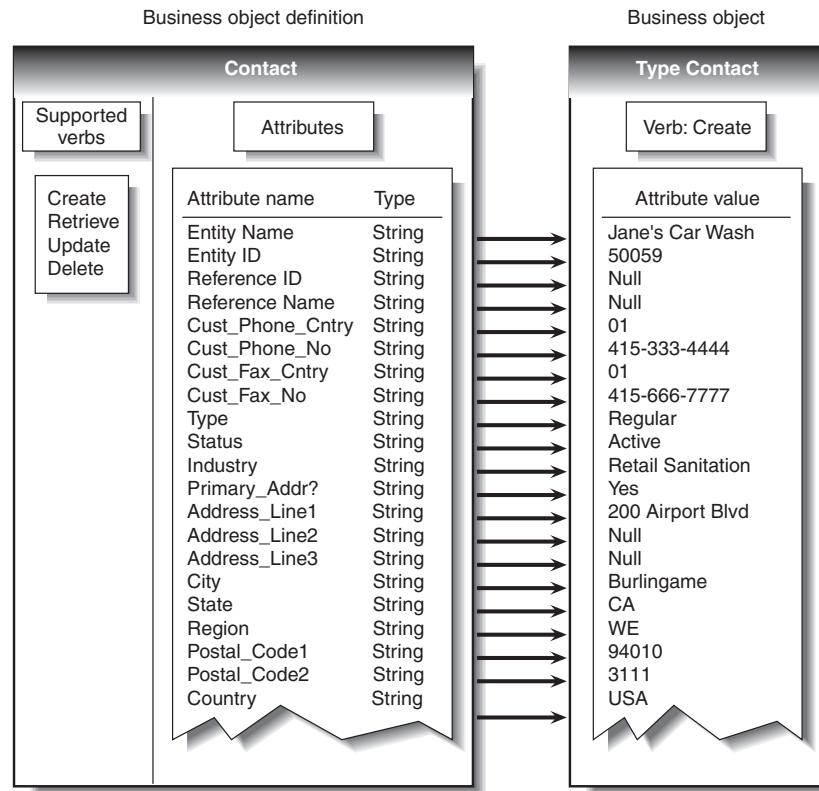


Figure 9. Business object definition and business object.

Components of a business object definition

In simplified terms, a business object is characterized by its type, its attribute values, and its verbs.

Overall, a business object definition is identified by its name. The name indicates the business object definition type, such as Customer, VantiveCase, or Invoice. A business object can also have application-specific information (metadata) that helps the application-specific component process it. All business objects also contain attributes and verbs, as the next sections describe.

Attributes

Attributes in a business object definition describe the values connected with the entity, such as Last Name, Employee ID, Case Number, Amount, or Date Initiated. At runtime, attributes are filled in with actual data.

For example, an Employee business object definition might contain attributes for the employee's name, address, employee ID, and other relevant information. The attributes of a business object are analogous to the fields of a form or columns in a database table.

An attribute can also refer to a child business object or to an array of child business objects, such as an array of line items in a contract or part references in an invoice.

ObjectEventId attribute: The ObjectEventId attribute is a required attribute and is the last attribute in every business object.

When a connector publishes an event, it uses the ObjectEventId attribute of the business object definition to store a unique value that identifies the specific business object instance that is being created. In some implementations, the value of the ObjectEventId attribute can be generated by the business integration system and used to identify and track the flow of the specific event through the system.

Basic and compound attribute types: If an attribute's type is a basic data type, such as String, Boolean, Double, Float, or Integer, the attribute value is a discrete piece of data, such as the value of a field in a database. Examples include LastName, CustomerID, PartNumber, AssignedTo, and Price.

If an attribute's type is the name of another business object definition (a compound type), the attribute value is a child business object or an array of child business objects. Examples include Customer, Contract, and Oracle_Contact.

Attribute properties: A number of **properties** define the value that the attribute represents. Without showing all possible properties, Figure 10 illustrates the place of attribute properties in a business object definition.

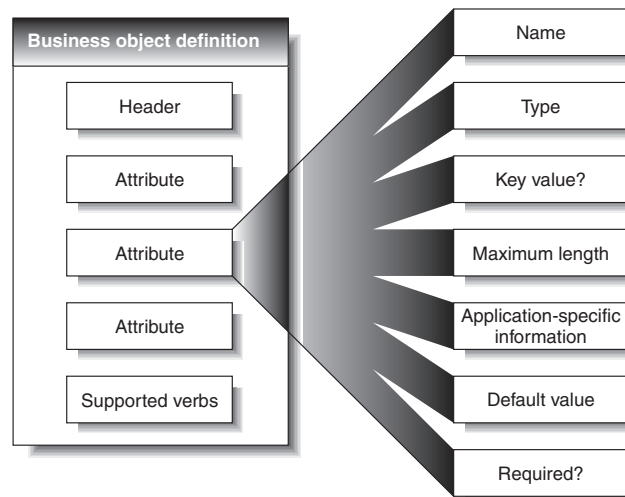


Figure 10. Attribute properties.

The set of properties for a particular attribute depends on whether the attribute type is basic or compound; that is, an attribute's properties differ depending on whether the attribute refers to a single unit of data or to a child business object.

Verbs

Verbs indicate actions on the data in the business object. A business object definition contains a list of verbs; a business object contains only one verb.

The most common verbs associated with business object definitions are Create, Retrieve, Update, and Delete.

The meaning of a verb differs according to the role of the business object. The verb can describe an application event, make a call, make a request, or identify the result of a previous request.

Note: Some applications do not support requests for hard deletes. For such applications, the business integration system performs the equivalent logical deletion, which is usually an update to inactive status. Furthermore, even if

an application supports hard deletes, you can configure the business integration system so that it converts Delete verbs to Update verbs when sending requests to that application.

A closer look at business objects

A business object contains the data that an application-specific component moves into or out of a particular application. Therefore, each business object definition reflects the application's data model and the application-specific component's access method.

Even when two application-specific business objects refer to similar application entities, differences appear in the way that attributes are organized and in the application-specific information for them.

Attribute organization

Applications often organize the same information in different ways. For example, Application A stores a telephone number and fax number for a contact in four fields, but Application B stores the same numbers in two fields.

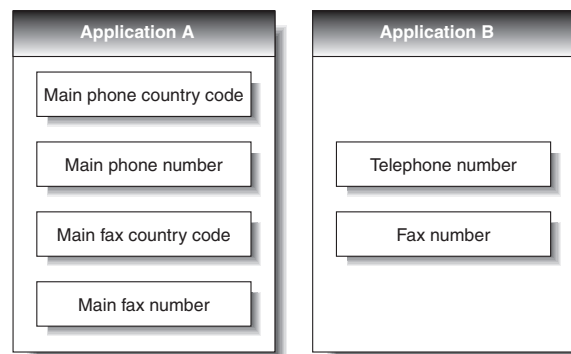


Figure 11. Telephone data in two applications.

The business object definitions for the Application A business object and the Application B business object have different attributes to reflect this difference.

Application-specific information

Business objects also differ because each can optionally contain built-in processing instructions for its application-specific component. Referred to as application-specific information (or metadata), it can consist of any information that the application-specific component needs to process the business object.

A business object definition can have application-specific information that applies to the entire business object, to each attribute, and to each verb. At each place where application-specific information appears in a business object definition, it provides information that the connector uses in its interactions with the application.

Application-specific information for a business object

Application-specific information for the business object provides information that the application-specific component uses when processing the business object as a whole.

Application-specific information for an attribute

Often, application-specific information that applies to an attribute identifies the attribute value's location in the application. The application-specific component uses this identifier when building API calls to the application to retrieve or enter the attribute value.

Application-specific information takes different forms for different applications. Sometimes the application-specific component can reference the attribute location by means of the application's form and field names; other times the reference is more complex.

Table 2 provides examples of parameters that might be included in an attribute's application-specific information. These parameters would be relevant only to a business object that represents data in a database table.

Table 2. Example name-value parameters for attribute application-specific information

Parameter	Description
TN= <i>TableName</i>	The name of the database table
CN= <i>col_name</i>	The name of the database column for this attribute.
FK=[<i>..</i>]fk_attributeName]	The value of the Foreign Key property defines a parent/child relationship.
UID=AUTO	This parameter notifies the connector to generate the unique ID for the business object and load the value in this attribute.
CA= <i>set_attr_name</i>	The Copy Attribute property instructs the connector to copy the value of one attribute into another. If <i>set_attr_name</i> is set to the name of another attribute within the current individual business object, the connector uses the value of the specified attribute to set the value of this attribute before it adds the business object to the database during a Create operation.
OB=[ASC DESC]	If a value is specified for the Order By parameter and the attribute is in a child business object, the connector uses the value of the attribute in the ORDER BY clause of retrieval queries to determine whether to retrieve the child business object in ascending order or descending order.
UNVL= <i>value</i>	Specifies the value the connector uses to represent a null when it retrieves a business object with null-valued attributes.

A single attribute's application-specific information might combine several of the example parameters listed above. This example uses semicolon (;) delimiters to separate the parameters:

```
TN=LineItems;CN=POid;FK=..PO_ID
```

The application-specific information in this example specifies the name of the table, the name of the column, and that the current attribute is a foreign key that links the child business object to its parent.

In exceptional cases, application-specific information for attributes is unnecessary. For example, some applications provide very direct and easy to use designations for units of data. Imagine that an application identifies sample fields as Table 3 on page 21 illustrates.

Table 3. Sample application identifiers

Attribute	Application's identifier for the field containing the value
Customer ID	XCustomerID
Customer name	XCustomerName
Status	XStatus
Industry	XIndustry

In the example that Table 3 illustrates, it is easy for the application-specific component to associate an attribute with its identifier in the application because the rules for conversion are so regular: add the X or subtract the X. Therefore, the attributes in business objects for this application may not need application-specific information.

Application-specific information for verbs

A business object definition can include application-specific information for each verb that it supports. The application-specific information tells the application-specific component how to process the business object when that verb is active.

Ways to create or modify business object definitions

Each connector requires a set of business object definitions to define the data that is to be communicated to other applications. When the application-specific component is required to send data to the integration broker, it instantiates a new business object from one of the business object definitions it supports. One step in the process of configuring a connector is to select the business object definitions to be supported. First, however, you need to create or otherwise generate business object definitions for the application.

Creating business object definitions

There are several ways to construct or obtain business object definitions for an application.

- If an object discovery agent (ODA) exists for your application, you can use it to build business object definitions. An ODA examines the structure and organization of the application's stored data and constructs business object definitions based on what it finds. If an ODA does not exist for your application, you can use the object discovery agent development kit (ODK) to build an ODA.
- You can use the Business Object Designer tool to create business object definitions, either by modifying those generated by an ODA or by constructing them from scratch.

Other resources

The *Business Object Development Guide* provides detailed information about creating business object definitions.

In addition, many adapters include sample business objects. If samples are included, they are located in the product directory under:

Windows:

`\connectors\ConnName\Samples`

UNIX:

/connectors/ConnName/Samples.

Modifying business object definitions

You might need to modify a business object definition for several reasons - to capture additional application data, to stop collecting data found to be unnecessary, or to respond to changes to another application. The Business Object Designer tool, described in the *Business Object Development Guide*, is the most convenient way to make these modifications.

Chapter 3. Connectors

A connector mediates between an application and the integration broker on a local network. It can be specific to an application—such as SAP R/3, version 4—or to a data format or protocol, such as XML or WebSphere MQ. It consists of an application-specific component and a connector framework.

All connectors share certain common behaviors, differing only in the manner in which they interact with applications and with business objects. This chapter is an introduction to both the common behavior of connectors and to the areas in which they differ. It includes the following sections:

- “Connector startup”
- “Event notification”
- “Request processing” on page 29
- “Guaranteed event delivery” on page 29
- “Connector configuration” on page 34
- “Connector development” on page 34

In some environments, connectors are “black boxes”; you can simply install, configure, administer, and use a connector without much concern for its internals. If you need to create a custom connector, however, you need more detailed knowledge of connector behavior. For information on creating or modifying a connector, refer to the *Connector Development Guide for Java* or the *Connector Development Guide for C++*.

Connector startup

A connector must be explicitly started using a startup script which can be invoked from the command line or from a Windows shortcut.

Each connector has a local repository, which holds the connector’s configuration file and a separate XML schema document for each business object definition. The repository is a directory in the local file system where the application-specific component is installed.

During startup, the connector does the following:

1. Loads the supported business object definitions and configuration properties from its local repository.
2. Connects to the application.

Note: An integration broker need not be running when you start a connector. However, no data can be transferred until the integration broker is active.

Event notification

Note: “Event notification” refers to the processing of events that originate from the connector’s event notification mechanism. In the Agent Request interaction pattern, the processing of the message that contains such an event requires a response from the integration broker; this is sometimes referred to as “synchronous event delivery” or “synchronous event processing.” In the

Agent Delivery interaction pattern, a response from the integration broker is not required; this is sometimes referred to as “asynchronous request processing”.

A connector whose application provides triggering events must learn about those events and send the associated data to the integration broker. Figure 12 illustrates a connector’s interactions with respect to event notification.

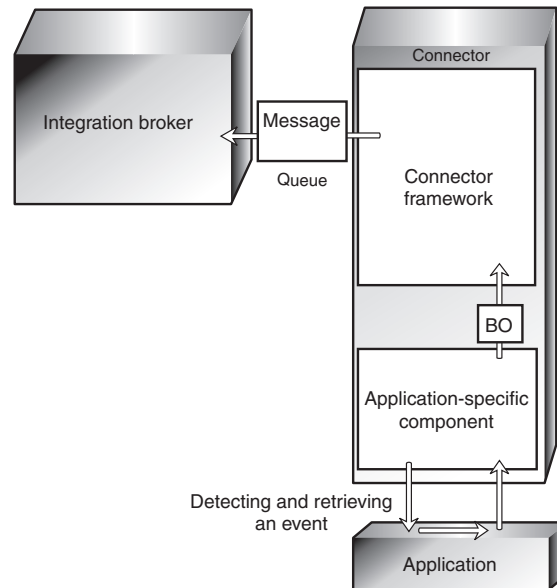


Figure 12. Event notification in a connector.

The ways in which application-specific components detect and retrieve events differ from one connector to another. However, the way in which application-specific components send events to the connector framework, and the way in which the connector framework deliver those events to the integration broker, is standard across all connectors.

The following subsections describe general concepts regarding the operation of most connectors, including:

- How connectors use application event-notification mechanisms
- How connectors detect and process events.

This discussion is not intended to describe the specific implementation of any particular connector.

Setting up the application’s event-notification mechanism

To a connector, an application event is any operation that affects the data of an application entity that is associated with a WebSphere Business Integration adapters business object definition. There are other types of events in applications; for example, a mouse click is an event to an application’s window system or forms interface. The connector, however, is interested only in a pre-defined subset of the data-level events that create, update, delete, or otherwise affect the content of the application’s data store.

Some applications explicitly trap and report events, providing user-friendly event management and configurable event text. Other applications, without a concept of

discrete, reportable events, might silently update their databases when something happens. WebSphere Business Integration adapters provide connectors for both types of applications.

For most connectors, some application configuration is needed to set up an **event-notification mechanism** for the connector's use. An event-notification mechanism maintains an ordered list of operations that take place in the application. It might have the physical form of an application event queue, an e-mail inbox, or a database table.

What types of event-notification mechanisms do connectors use? The next sections illustrate some general approaches.

When applications have event support

If an application is event-based, it probably has an event-notification interface for use by client applications such as connectors. The application might also permit you to configure the text of the event report. For such applications, setting up the connector's event-notification mechanism is a normal application setup task.

For example, imagine that an application lets you install a script that executes when a particular type of event occurs and that the script can place a notification in an event inbox. To install the connector for that application, you create a user account for the connector, write or obtain scripts for handling the events you want to track, install the scripts, specify the type of event that triggers each script, and create the inbox. When you are done, the application-specific component periodically retrieves the inbox contents to check for new events.

Figure 13 illustrates an application configuration that includes an event inbox.

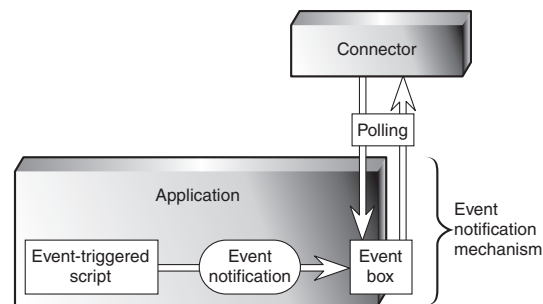


Figure 13. Example: Using an event inbox for event notification.

Another application might have an internal workflow system that can generate e-mail messages or write to an event queue when a particular operation occurs. Figure 14 on page 26 illustrates an application that has its own business object repository where business objects and events are defined. In the figure, Customer is a business object and Create, Delete, and Update are the types of events associated with it.

When a business object event such as Customer.Update takes place, the event is sent to the workflow system, which places an entry in an event table in the application database.

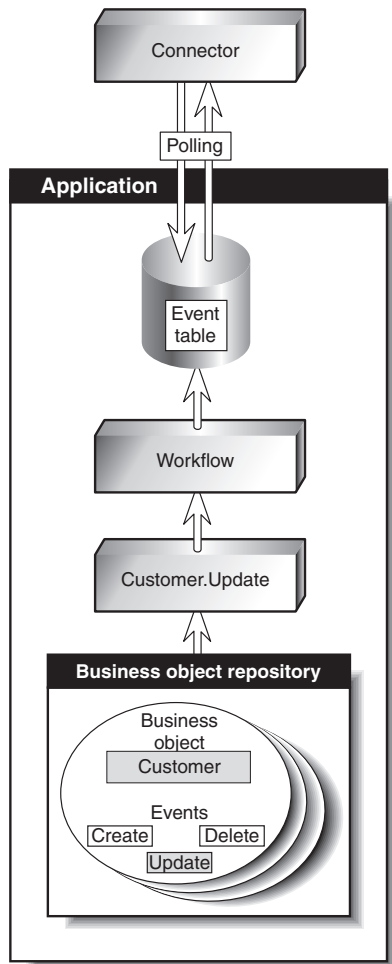


Figure 14. Example: Using application workflow for event notification.

When applications lack event support

The preferred method for a connector to interact with application events is through the application's API, which provides a framework that enforces the application's data model and logic. However, some application APIs do not provide native support for event notification.

One way that a connector can receive event notifications from such an application is to interact with the application database. For example, you can set up a trigger on an Employee table that detects updates to the rows. When an update occurs, the trigger inserts information about the update into a table, created when you deploy the connector. Each new row that appears in the event table represents an event notification. The connector can use SQL queries to retrieve new events from the table.

Figure 15 on page 27 illustrates this approach.

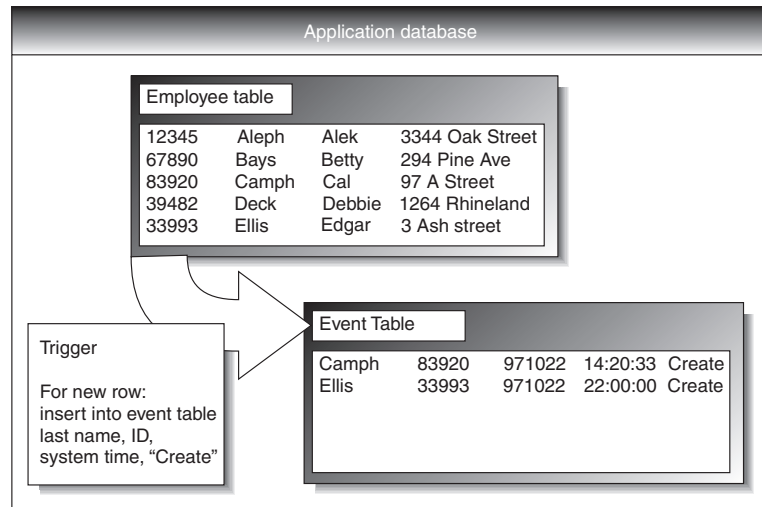


Figure 15. Example: Using the database for event notification.

In Figure 15, the application database has a trigger on the creation of records in the Employee Table. Each time the application inserts a new record, the trigger creates a row in the event table. The row contains the key values of the new employee record (last name and employee ID), the system time, and the event type, Create.

Detecting an event

A connector's application-specific component learns about application events through its event-notification mechanism, the most common of which is polling for new events in the event store. The polling method is specific to the application, based on the event-notification mechanism that the connector uses.

Polling is configurable. When you use the Connector Configurator tool to configure a connector, you can:

- Adjust the frequency with which the application-specific component polls the application
- Specify the hours during which the application-specific component polls the application.

For most connectors, you can also specify the number of events to be processed per poll call.

An application-specific component need only poll an application if another application is interested in that application's events. If a particular application is not the source of events, you can stop the application-specific component from polling by setting its polling frequency to "no" using the Connector Configurator. To learn more about the Connector Configurator, refer to either of the Connector Development Guides or to the adapter user guide for the adapter you are deploying.

Processing an event

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

- Associates the application event with a business object definition and creates an instance of that business object
- Sets the verb and key value attributes in the business object

- Retrieves application data and populates the business object’s attributes
- Sends the business object to the connector framework
- Archives the event (optional).

Associating an application event with a business object definition

When an application-specific component retrieves an event, it must determine which business object definition and verb represent the event.

The application-specific component uses the event text to associate the event with a business object definition and verb, as Table 4 shows.

Table 4. Event text and business object formation

Type of data in the application event	Examples	Use
Application entity type	Customer, Part, Item	Determining the associated business object definition
Operation that occurred	Create, Update, Delete	Determining the active verb of the business object

For example, a connector can associate the following event text with an Employee.Create business object:

```
1997.10.19.12:50.22 employee created lname="como" id="101961"
```

From left to right, the event text consists of:

- A time stamp that helps to uniquely identify the event
- The application entity “employee”
- The operation “created”
- The employee’s last name and ID, unique identifiers (key values) with which the application-specific component can retrieve the rest of the employee information.

Note that this example is simple; other types of event text might require more processing by the application-specific component.

Building an application-specific business object

If the connector’s been configured to support the business object definition for the event, the application-specific component builds a business object, uses a key value to retrieve application data, and fills in the business object with the application data. “Business object construction and deconstruction” on page 31 describes the process of building a business object.

Sending the application-specific business object to the connector framework

The application-specific component sends the business object to the connector framework without needing to know the identity of the business process that will receive the information carried in the business object.

Archiving events

Application event archives are useful for troubleshooting and record-keeping. An event archive contains status information about each event, such as:

- Successfully sent to the integration broker
- Processing failed

If an application provides an event archiving feature, the connector generally uses it. A connector for an application that does not support event archiving might have its own event archive. For example, if a connector's event-notification mechanism is like the database mechanism illustrated in Figure 15 on page 27, a database trigger could copy deleted events to an archive table. that you create when you deploy the connector.

Guaranteed event delivery

Guaranteed event delivery ensures that critical events, such as financial transactions, are processed correctly, regardless of any service interruptions that might occur. This feature enables the connector framework to guarantee that each event is detected and transmitted only once from the source connector's event store to the destination connector's request queue.

Without guaranteed event delivery, a small window of possible failure exists between the time that the connector detects an event and the time all necessary processing has completed. If a failure occurs in this window, the event has been sent but its event record remains in the event store. When the connector restarts, it finds this event record still in the event store and treats it as a new event, causing the event to be sent twice.

For more information about guaranteed event delivery and to learn how to enable it for the adapter you are deploying, refer to the adapter's user guide.

Request processing

Note: In WebSphere Business Integration adapters documentation, "request processing" refers to the handling of a business object request message sent from the integration broker to a connector. The Hub Request interaction pattern is request processing in which the connector is required to return a result to the integration broker. The Hub One Way interaction pattern is request processing in which the connector is not required to return a result to the integration broker.

When a process tells it to do so, the integration broker sends requests, in the form of WebSphere MQ messages, to a connector.

In a typical scenario, the integration broker first receives an event notification business object message from the connector for a source application, and as a result of processing that event, it then sends a request business object message to the connector for a destination application.

A request can be made to the destination application to do either of the following:

- To retrieve business data and return it to the integration broker.
- To update the application's data store.

For example, the integration broker might send a connector a request message to delete a contract, update a part, or create a customer.

When the connector framework receives an integration broker's request, it converts the message into a suitable business object and forwards it to the application-specific component. For example, if the integration broker sends a request to delete a contract, the application-specific component receives the request in the form of a `Customer.Delete` business object. The application-specific

component translates the business object into an application request—typically a set of calls to the API—and then returns the results, if needed.

Figure 16 illustrates a connector's interactions with respect to handling messages from the integration broker in a Hub Request interaction pattern.

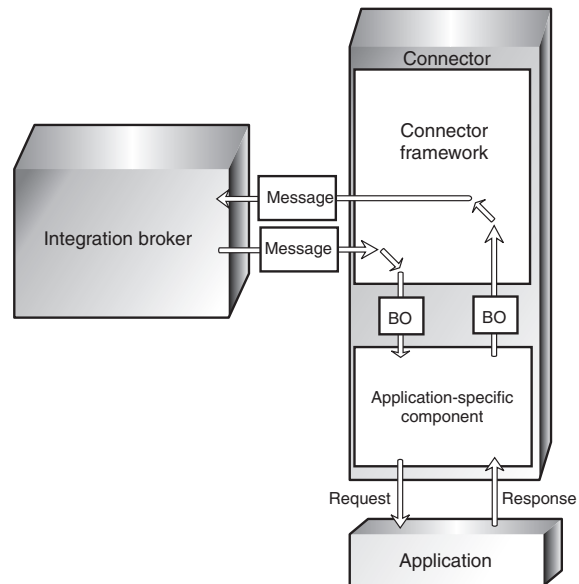


Figure 16. Connector interactions for request handling.

When an application-specific component receives a request, it determines how to process the request based on three types of information:

- The verb of the business object
- Metadata that is contained in the business object definition itself and used in the construction and deconstruction of the business object
- Application-specific information for the verb.

These factors are described in the topics that follow.

Verb-based processing

A connector's application-specific component responds to the Create, Retrieve, Update, or Delete verb in a request according to the logic and API of its application. The application-specific components of different connectors might handle the same type of request differently, although the result is logically the same.

For some connectors, only one method is required for performing operations on a business object, regardless of what verb the request contains. But for many connectors, each verb requires a different method.

When an application-specific component receives a request, it invokes the method in the application that matches the business object's active verb. For example, when an application-specific component receives an AppAEmployee.Update business object, it invokes the Update method on the AppAEmployee object. The Update method interacts with the application in order to perform the update.

Figure 17 illustrates some verb handling methods.

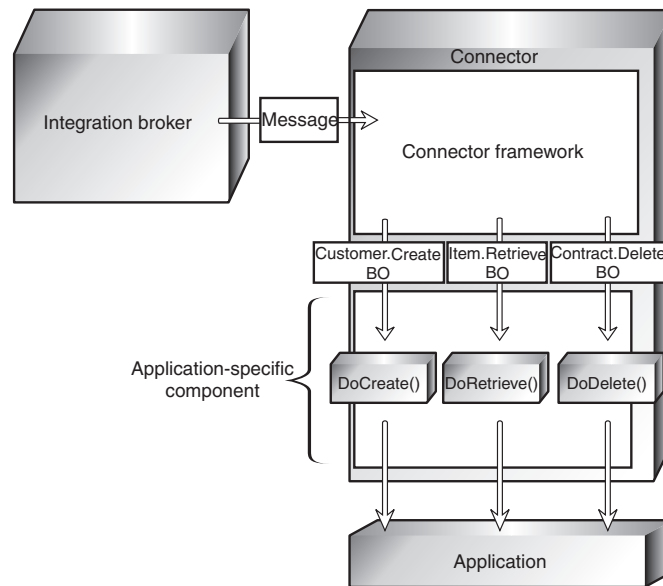


Figure 17. The processing of requests.

When the connector in Figure 17 receives a Customer.Create, Item.Retrieve, or Contract.Delete request, it invokes its DoCreate(), DoRetrieve(), or DoDelete() method, respectively. Each method builds the appropriate commands for the application to perform the specified operation.

Business object construction and deconstruction

A connector's application-specific component accomplishes its event notification and request-handling tasks by constructing and deconstructing business objects:

- When an application-specific component detects an event that it must send to the integration broker, it constructs a business object that represents the event.
- When an application-specific component receives a business object that represents a request from the integration broker, it deconstructs the business object to create an application request.

Business object metadata and connector actions

A connector's transformation of an application event to a business object and from a business object to an application request is driven by data definitions (**metadata**) that are defined when a business object is designed.

Application-specific components and business object metadata are designed to work together. The design of an application-specific component and its business objects is analogous to the design of a computer device in which certain functions can be implemented by either the software or hardware. The developer considers performance, extensibility, and other issues to decide where to implement key features.

Business object definitions include properties that specify the types, sizes, and default values for attributes. They also include application-specific properties that contain instructions to the application-specific component on how to process the business object.

Table 5. Example name-value parameters for application-specific information associated with attributes.

Parameter	Description
TN= <i>TableName</i>	The name of the database table
CN= <i>col_name</i>	The name of the database column for this attribute.
FK=[<i>..</i>]fk_attributeName]	The value of the Foreign Key property defines a parent/child relationship.
UID=AUTO	This parameter notifies the connector to generate the unique ID for the business object and load the value in this attribute.
CA= <i>set_attr_name</i>	The Copy Attribute property instructs the connector to copy the value of one attribute into another. If <i>set_attr_name</i> is set to the name of another attribute within the current individual business object, the connector uses the value of the specified attribute to set the value of this attribute before it adds the business object to the database during a Create operation.

When processing a business object, the application-specific component reads the definition and uses the application-specific information to build an application request. For more information on business objects, see the *Business Object Development Guide*.

Because the application-specific component is metadata-driven, its actions are controlled by application-specific information and other metadata in the business object definition. It does not have hard-coded instructions for each type of supported business object. Being metadata-driven gives the application-specific component the flexibility to automatically support any new or changed business object definition as long as the corresponding application data can be accurately described by the connector's metadata syntax.

An example of business object construction

The following process describes how an application-specific component creates a business object from its definition:

1. Obtains the business object definition from its local repository and uses it to create a business object instance.
2. The application-specific component loops through the business object instance attribute by attribute, using application-specific information to prepare an API call or build a query to obtain the application entity.
3. The application-specific component sends the request to the application and retrieves the results.
4. The application-specific component loops through the results, using the value of `AppSpecificInfo` to determine which retrieved value represents each business object attribute.

Figure 18 on page 33 is an example of an application-specific component that is building a business object from the definition. The application-specific component has retrieved an application event involving an item whose key value, the item number, is 123. The application-specific component must build an Item business object from the business object definition, which contains four attributes: Group, Description, Price, and ItemNum.

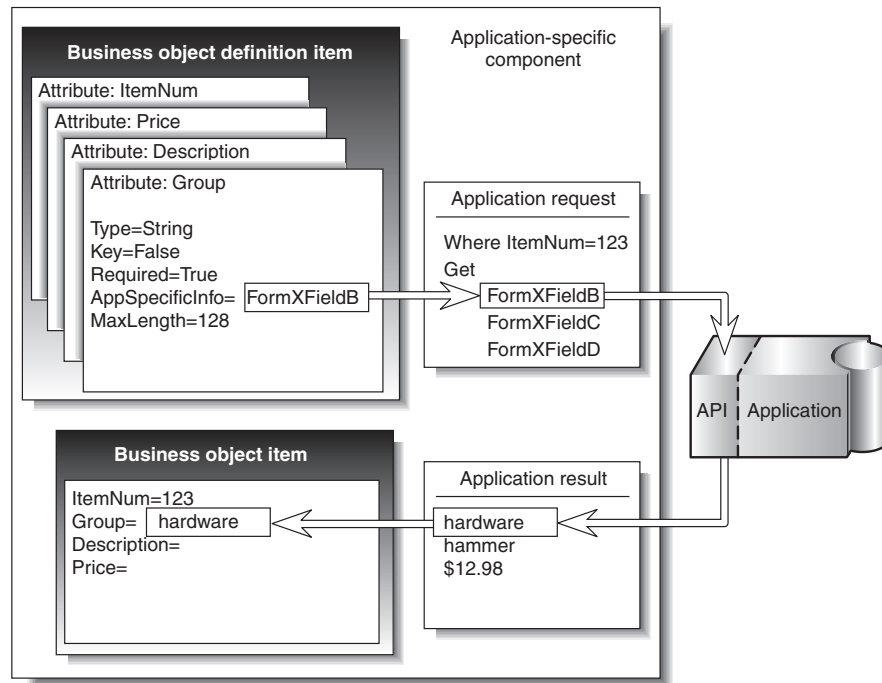


Figure 18. Building a business object in a connector.

Using the item number, 123, to identify the item, the application-specific component retrieves the values of the remaining attributes. Application-specific information identifies the form and field identifier for the required data.

For example, FormXFieldB identifies Group data. The application-specific component requests the value of Field B in Form X for item ID 123. The application-specific component then uses the returned value, "hardware," to fill in the value of the business object's Group attribute.

The process of deconstruction works in the opposite way. The application-specific component uses the business object definition to determine how to make an application request from the data contained in the business object that it received.

Application-specific information for verbs

Each verb in the business object definition can have application-specific information associated with it. The content of the verb's application-specific information is unique to the particular connector. The application-specific information for the verb provides the connector's application-specific component with additional instructions for processing the business object.

For example, application-specific information for the Retrieve verb in a business object definition might supply special input arguments to the Retrieve method in that application-specific component.

As an example, suppose that the MyApp application has three forms in which information about InventoryItem appears:

- InventoryItem-New
- InventoryItem-Change
- InventoryItem-Remove.

When the application-specific component for MyApp performs an operation on an inventory item, it must reference the correct form for that operation. In the InventoryItem business object definition, the application-specific information for the verb can be used to store the form name.

The combination of verb-specific methods and application-specific input to those methods gives an application-specific component unique instructions for processing.

Connector configuration

Before a connector can be used, you must use the Connector Configurator to define a configuration file that contains:

- The business objects that the connector supports
- The configuration properties for the connector.

There are two types of connector configuration properties, **standard properties** and **application-specific properties**.

Standard properties, which apply to all connectors, specify information such as:

- The integration broker used with the connector
- The location of the connector's local repository
- The name of the queue manager that manages the queues used by the connector.

Application-specific properties specify values that a particular application-specific component needs to establish a session with the application. They also direct certain aspects of the application-specific components processing behavior. Here are examples of application-specific properties for various connectors:

- The name or IP address of the machine running the application
- The name of the application database
- The login user ID and password the connector needs to use to access the application
- The name of the event inbox
- The number of events to be retrieved per polling event.

Some connector configuration properties can also be set at the command line when you start up the connector. Properties set on the command line override the values set in the connector's configuration file.

For more information on configuring a connector, refer to "Configuring the connector" on page 57. To learn more about starting up a connector, see "Starting a connector" on page 105.

Connector development

To modify or create a connector, you use class libraries and samples to create an application-specific component. You then use the Connector Configurator to create and modify business object definitions and create connector repository definitions.

Connector development involves defining the relationship between the application-specific component and a particular application. The actual coding of a application-specific component is usually a fairly straightforward process. The most challenging tasks are:

- Designing the application's event-notification method
- Defining business object definitions
- Defining the relationship between the business objects and the application objects.

For detailed information on connector architecture, modifications, and development, refer to the *Connector Development Guide for Java* or the *Connector Development Guide for C++*.

The next chapter, Chapter 4, "Data transport and the integration broker," on page 37, provides a more detailed look at the mechanisms and protocols used to transfer information among the business integration system's components.

Chapter 4. Data transport and the integration broker

WebSphere Business Integration adapters implemented with WebSphere Application Server as the integration broker use WebSphere MQ queues and Java Messaging Service (JMS) software as the communication transport mechanism between the connector framework and the integration broker. Connectors read messages from and write messages to pre-defined queues, managed by the WebSphere MQ Queue manager.

This chapter describes in detail the messaging interfaces and protocols used to exchange data among applications in the WebSphere business integration system.

The role of the integration broker

The integration broker both routes messages between application connectors and performs processing on the messages. Specifically, the integration broker:

- Receives a message from the outbound queue of a connector.
- Performs processing on that message.
- As a result of that processing, sends a messages to the inbound queues of one or more connectors.

Communication between the integration broker and a connector can be **asynchronous** or **synchronous**.

Asynchronous data transport

Programs that use asynchronous messaging transport need not establish connections or wait for messages (discrete units of data); each program sends and receives messages by interacting asynchronously with the messaging service. The messaging service provides guaranteed delivery, storing the message if the destination program is unavailable and retrying until it is available.

Synchronous data transport

Programs that use synchronous messaging transport post request messages to the integration broker on a synchronous request queue and receive response messages from the integration broker from a synchronous response queue. A correlation ID on the response message identifies the request message to which it is responding. Response messages generally consist of business object messages and a status indicator that shows whether the request was processed successfully.

Interfaces for message exchange

This section describes the messaging interfaces used by the connector framework and the integration broker to transmit messages and the information needed to process them.

Several distinct types of messages are exchanged within the business integration system. Essential information needed to identify the different types of messages, as well as process and route them correctly, is stored in the message header and the message descriptor of each message. Message flows you create for your business

integration system use the information presented below to recognize and correctly manage messages they are called upon to process.

The following types of messages are passed:

- Event delivery messages are sent by the connector framework to the integration broker to notify the broker of an event in the source application. In the WebSphere Application Server environment, such messages are commonly referred to as Agent Delivery. Agent delivery messages are written to the Delivery Queue (see "Required Types of Queues," below) by the connector framework, and are read from the DeliveryQueue by the integration broker.
- Request messages are exchanged between the connector framework and the integration broker to convey a request for data.

When the integration broker sends a request to the connector framework and expects a response, the exchange is referred to as a HubRequest. In a HubRequest, the request from the integration broker is written to the RequestQueue, where it is read by the connector framework.

A request from the connector framework to the integration broker is referred to as an AgentRequest. An AgentRequest is written by the connector to the SynchronousRequestQueue, where it is read by the integration broker.

In requests from either the integration broker or the connector framework, the JMSReplyto property in the message header of the request determines whether a response is expected, and specifies the queue to which the response should be sent.

- Response messages are exchanged between the connector framework and the integration broker to reply to a request for data. When the connector framework responds to a request from the integration broker, the response message is written to a ResponseQueue. When the integration broker responds to a request from the connector framework, the response message is written to the SynchronousResponseQueue.
- One-way messages are sent from the integration broker to the connector framework. In the WebSphere Application Server environment, such messages are commonly referred to as HubOneWay. A HubOneWay message is written to a RequestQueue by the integration broker and is read from the RequestQueue by the connector framework.
- Administrative messages are exchanged between the connector framework and the integration broker to convey administrative commands.

Message formats

Messages exchanged between the connector framework and the integration broker are formatted by the data handler, based on:

- The WireFormat standard property in the connector's configuration file
- The XML schema detailing the message body format
- The content of the message: a business object or an administrative message
- The origin and destination of the message.

Each message contains three components: a message descriptor (MQMD), a message header (MQRFH2), and a message body.

Message descriptor

The WebSphere MQ message descriptor (MQMD) contains the message ID and includes information needed for processing the message.

Message header

The MQRFH2 message header carries JMS-specific data that is associated with the message content. It can also carry additional information that is not directly associated with JMS.

Message body

The message body is formatted as specified by the XML schema specified for the message. In order for the data handler to find and use the correct XML schema for formatting a message, the following three names must be the same:

- The name of the XML schema stored in the connector's repository
- The name of the XML schema imported into the integration broker's message repository and saved as a message set definition.
- The value of `messagetype` in the message's MQRFH2 message header.

The message formats and the settings for particular properties for the different types of messages exchanged by the connector framework and the integration broker are listed in Appendix A, "WebSphere MQ message formats," on page 123.

Message queues

The WebSphere MQ queues that need to be defined and configured for use with the connector are described below.

Required types of queues

Separate sets of WebSphere MQ message queues are used for transporting business object messages and administrative messages between the connector framework and WebSphere Application Server. For descriptions of these queues, see "Creating the WebSphere MQ queues" on page 54.

Queue manager

The connector uses a single queue manager to manage all of its interactions with queues. The standard properties in the connector's configuration file contain the queue manager information needed by the connector at startup. The connector uses this information to establish a connection to the queue manager it will use to communicate with the integration broker.

The WebSphere business integration system supports several queue managers and queue configurations. The connector can communicate with the queue manager in any of the following modes:

- **Bindings mode:** The integration broker and the connector communicate directly with the queue manager, without using a TCP/IP connection. The queue manager and the connector must be on the same machine and must use the same queue manager. This is the default mode.
- **Bindings mode with remote queue definitions:** If the integration broker and the connector are installed on separate machines, with each machine running its own queue manager, the connector and the integration broker can still communicate with their respective queue managers using bindings mode but remote queue definitions are also needed.
- **Client mode:** Communication occurs through a client connection that uses TCP/IP as its underlying transport. If the queue manager and the connector are on the different machines, the connector is limited to using client mode.

For more information

To learn more about WebSphere MQ messages see *WebSphere MQ: Using Java*. To learn more about WebSphere MQ queues, see *WebSphere MQ: Intercommunication* and *WebSphere MQ: Script Command (MQSC) Reference*.

Part 2. Deployment and administration

Chapter 5. Overview: Implementing WebSphere Application Server as an integration broker

WebSphere Application Server uses J2EE technology to fulfill the role of an integration broker, implementing data exchanges between disparate applications across a network or across the Internet.

The WebSphere Application Server environment supports four types of operations for the exchange of business data. These types are sometimes referred to as "interaction patterns." The following table describes each interaction pattern and the WebSphere MQ queues that are used to implement it:

Interaction pattern	Description	Queues used
Agent Request	Connector sends synchronous request message to WebSphere Application Server, reporting the occurrence of an event in the application, and waits for a response. (In the documentation for individual adapters, this is commonly referred to as <i>Synchronous Event Delivery</i> .)	SynchronousRequestQueue for request to broker. SynchronousResponseQueue for response to connector.
Agent Delivery	Connector sends asynchronous message to broker, reporting the occurrence of an event in the application. Connector does not wait for a response. (In the documentation for individual adapters, this is commonly referred to as <i>Asynchronous Event Delivery</i> .)	DeliveryQueue
Hub Request	Broker sends synchronous request message to a connector and waits for a response. (In the documentation for individual adapters, this is commonly referred to as <i>Synchronous Request Processing</i> .)	RequestQueue for message from broker to connector. ResponseQueue for response from connector to broker.
Hub One Way	Broker sends asynchronous request message to the connector and does not wait for a response. (In the documentation for individual adapters, this is commonly referred to as an <i>Asynchronous Request Processing</i> .)	RequestQueue

At a high level, the tasks necessary for implementing WebSphere Application Server as the integration broker for these exchanges are the following:

- Design and development of the integration components

Integration components are the WebSphere Business Integration adapter connectors that interact with applications and the WebSphere Business Integration adapter business objects that contain the data that is exchanged. Part

One of this guide introduces the necessary concepts for understanding the integration components, and provides links to more detailed documentation for the design and development of WebSphere Business Integration adapter connectors and business objects.

- Installation and configuration of the integration components

Information for installing the connector is provided later in this guide in Chapter 6, "Installing WebSphere Business Integration adapters," on page 45 and in additional documents referenced from that chapter. Information for configuring the connector and its business objects, including the tasks of setting up queues and using Connector Configurator, is provided later in this guide.

- Deploying configured components to WebSphere Application Server

This includes the use of System Manager to create user projects from configured components, and the use of WebSphere Studio Application Developer to create service projects that can be deployed to WebSphere Application Server. These tasks are described later in this guide.

Chapter 6. Installing WebSphere Business Integration adapters

This chapter describes how to install WebSphere Business Integration adapters and its supporting software using a WebSphere Application Server as the integration broker.

If you are upgrading WebSphere Business Integration adapters from a previous version, see Appendix F, “Upgrading WebSphere Business Integration adapters,” on page 169 for instructions.

This chapter describes installation on both Windows and UNIX operating systems. You need only refer to the section that is appropriate for your operating system, as listed below:

- “Installing for Windows systems”
- “Installing for UNIX systems” on page 47

This chapter contains information specific to installing the supporting software (JDK and WebSphere MQ) on the integration broker. A separate document, the *Installation Guide for WebSphere Business Integration Adapters*, contains all the details necessary for installation of WebSphere Business Integration adapters. The sections of this chapter that describe installation for either Windows or UNIX systems refer you to the *Installation Guide for WebSphere Business Integration Adapters* at the appropriate point in the installation process.

Installing for Windows systems

This section includes the following topics:

- “Software Requirements”
- “Installing the JDK” on page 46
- “Installing WebSphere MQ” on page 46
- “Installing WebSphere Business Integration adapters” on page 46
- “Installing plug-ins for WebSphere Studio Application Developer Integration Edition” on page 46

Software Requirements

The WebSphere business integration system includes WebSphere Business Integration adapters and other components. WebSphere Business Integration adapters and some associated components are delivered on CD or through ESD (IBM Electronic Software Delivery). Other components used by the WebSphere business integration system must be obtained and installed separately.

Table 6 on page 46 lists the software requirements for the WebSphere business integration system.

Table 6. WebSphere business integration system software requirements for Windows systems

Software	Shipped with product
Windows 2000 (Professional, Server, or Advanced Server) with Service Pack 4. Note: Beginning with the release of WebSphere Business Integration Adapter Framework 2.4.0, adapters are no longer supported on Windows NT.	No
IBM WebSphere Application Server Enterprise Edition, V. 5.0.2, 5.1	No
WebSphere Studio Application Developer Integration Edition. V. 5.0.1, 5.1.1	No
Java Runtime Environment (JRE): IBM version 1.4.2	Yes
Java Development Kit - only needed for developing custom Java connectors. IBM JDK version 1.4.2	Yes
IBM WebSphere MQ version 5.3.0.5 (this is version 5.3 with CSD05).	No
Browser: An HTML browser such as Microsoft Internet Explorer or Netscape Navigator is required for viewing the HTML documents. For the exact versions supported, refer to the instructions that can be downloaded from linkend="WBIASET1034523">http://www.ibm.com/integration/wbiadapters/library/infocenter.	No

Installing the JDK

The JDK (Java Development Kit) is required only if you plan to develop custom Java connectors (the Java compiler is needed). A single CD contains JDKs for all supported platforms. For information on installing the JDK, see *Installing WebSphere Business Integration Adapters, Version 2.6*.

Installing WebSphere MQ

IBM WebSphere MQ is the messaging software that enables communication between the integration broker and the adapters.

Refer to the following WebSphere MQ publications for installation and configuration information:

- *WebSphere MQ: Quick Beginnings*
- *WebSphere MQ: System Administration*
- *WebSphere MQ: Intercommunication*

Note: You can browse or download these documents from IBM's Web site at: www.ibm.com/software/integration/mqfamily.

To configure WebSphere MQ to work with the integration broker, see "Configuring WebSphere MQ queues for WebSphere Application Server" on page 53..

Installing WebSphere Business Integration adapters

For information on installing WebSphere Business Integration adapter products, refer to the *Installing WebSphere Business Integration Adapters* guide located in the WebSphere Business Integration Adapters Infocenter at the following site:

<http://www.ibm.com/websphere/integration/wbiadapters/infocenter>

Installing plug-ins for WebSphere Studio Application Developer Integration Edition

To use WebSphere Application Server as the integration broker with WebSphere Business Integration adapters, you must add certain plug-ins to your existing

installation of WebSphere Studio Application Developer Integration Edition. The plug-ins are copied to your system when you run the installer for the WebSphere Business Integration Adapter Framework (see “Installing WebSphere Business Integration adapters” on page 46. The Adapter Framework installer copies the following plug-ins to the location that you specify:

```
\plugins\com.ibm.btools.adaptermonitor  
\plugins\com.ibm.btools.adaptermonitor.help  
\plugins\com.ibm.btools.csm  
\plugins\com.ibm.btools.csm.help  
\plugins\com.ibm.btools.itools.ae.settings  
\plugins\com.ibm.btools.itools.codeGen  
\plugins\com.ibm.btools.itools.cwconverter  
\plugins\com.ibm.btools.itools.datamanager  
\plugins\com.ibm.btools.itools.msg.importer  
\plugins\com.ibm.btools.itools.wsdlgen
```

In order for the installer to copy the plug-in files to the correct location on your system, you should already have WebSphere Studio Application Developer Integration Editions installed, before you run the Adapter Framework installer. For general instructions about installing WebSphere Studio Application Developer Integration Edition, consult the documentation that is available for that product through the Infocenter.

Installing for UNIX systems

This section includes the following topics:

- “Software Requirements”
- “Installing the JDK” on page 48
- “Installing WebSphere MQ” on page 48
- “Installing WebSphere Business Integration adapters” on page 49

Software Requirements

The WebSphere business integration system includes WebSphere Business Integration adapters and other components. WebSphere Business Integration adapters and some associated components are delivered on CD or through ESD (IBM Electronic Software Delivery). Other components used by the WebSphere business integration system must be obtained and installed separately.

Table 7 on page 48 lists the software requirements for the WebSphere business integration system.

Table 7. WebSphere business integration system software requirements for UNIX systems.

Software	Shipped with product
Operating system: <ul style="list-style-type: none"> • AIX - AIX 5L version 5.1 maintenance level 1 or 5.2 with maintenance level 4 • Solaris - Solaris 7.0 or 8.0 , with Patch Cluster released 7/23/03 • HP-UX- HP-UX 11.11 (11i,r=B.11.110306.4). June 2003 GOLDBASE 11i,r=B.11.110306.4 and June 2003 GOLDAPPS11i 11i,r=B.11.110306.4 bundles. 	No
IBM WebSphere Application Server Enterprise Edition, V. 5.0.2	No
Java Runtime Environment: <ul style="list-style-type: none"> • AIX - IBM JRE version 1.4.2. • Solaris: Sun JRE version 1.4.2. • HP-UX - Sun JRE version 1.4.2. • Linux - IBM JRE version 1.4.2 	Yes
Java Development Kit - only needed for developing custom Java connectors. <ul style="list-style-type: none"> • AIX: IBM JDK version 1.4.2. • Solaris: Sun JDK version 1.4.2. • HP-UX: Sun JDK version 1.4.2 • Linux - IBM JDK version 1.4.2 	No
WebSphere MQ version 5.3.0.5 (this is version 5.3 with CSD05).	No
Browser: An HTML browser such as Microsoft Internet Explorer or Netscape Navigator is required for viewing the HTML documents. For the exact versions supported, refer to the instructions that can be downloaded from linkend="WBIASET1034523">http://www.ibm.com/integration/wbiadapters/library/infocenter .	No

Installing the JDK

The JRE includes the Java Virtual Machine (JVM), which is needed to run WebSphere Business Integration adapters. However, it does *not* include development tools, such as JavaC (the Java compiler). If you do *not* plan to create custom connectors, use the JRE.

The JRE contains the runtime component of the Java software, which the WebSphere business integration system requires to execute. The JRE is included in WebSphere Business Integration Adapter Framework. Therefore, there is no need to install it separately.

If you plan to create custom connectors, install the full JDK. A single CD contains JDKs for all supported platforms. For information on installing the JDK, see *Installing WebSphere Business Integration Adapters, Version 2.6*.

Installing WebSphere MQ

IBM WebSphere MQ is the messaging software that enables communication between an integration broker and the adapters.

Refer to the following WebSphere MQ publications for installation and configuration information:

- *WebSphere MQ: Quick Beginnings*
- *WebSphere MQ: System Administration*
- *WebSphere MQ: Intercommunication*

Note: You can browse or download these documents from IBM's Web site at:
<http://www.ibm.com/software/integration/mqfamily>.

To configure WebSphere MQ to work with the integration broker, see "Configuring WebSphere MQ queues for WebSphere Application Server" on page 53..

Installing WebSphere Business Integration adapters

For information on installing WebSphere Business Integration adapter products, refer to the *Installing WebSphere Business Integration Adapters* guide located in the WebSphere Business Integration Adapters Infocenter at the following site:

<http://www.ibm.com/websphere/integration/wbiadapters/infocenter>

Chapter 7. Configuring the WebSphere business integration system

This chapter describes the tasks necessary for setting up a connector for use with WebSphere Application Server. These instructions assume the following:

- You have already installed WebSphere Application Server Enterprise, v. 5.0.2 or later and WebSphere Studio Application Developer Integration Edition, v. 5.0.1 or later.
- You have already installed the WebSphere Business Integration adapter and Adapter Framework, and other required software for the adapter (for example, a data handler which is required for some adapters but not others).
- You have installed MQSeries and other prerequisite software, as described in Prerequisites in the Installation chapter of this guide.
- You have already created or acquired a connector and installed it to your system. For instructions on installing a connector, see the documentation that accompanies the connector that you acquired, or see the Connector Development Guide if you have created your own connector.

This chapter explains how to configure WebSphere Application Server, business objects, and the connectors to work together as a business integration system. It includes the following sections:

- “Overview of configuration tasks”
- “Creating business object definitions” on page 52
- “Adding business objects to ICLs” on page 53
- “Configuring WebSphere MQ queues for WebSphere Application Server” on page 53
- “Enabling the application for use with the connector” on page 56
- “Configuring the connector” on page 57
- “Using Visual Test Connector to verify your interfaces” on page 64

Overview of configuration tasks

To configure the business integration system, you need to perform the following tasks:

1. Create business object definitions, and add them into your ICL projects in System Manager.
2. Configure WebSphere Application Server to support the connector by defining the necessary JMS queues for the data transactions you wish to implement. You will use the Administrative Console in WebSphere Application Server to set JNDI names for the queues, and you will later use Deployment Descriptor editor in the Business Integration perspective to map names.
3. If you have not already done so, generate business object definitions for use with the connector.
4. Add the business object definitions in System Manager
5. Configure your connector with its associated business objects, using Connector Configurator.
6. Configure standard and application-specific configuration properties for the connector, including the properties that specify queues.

7. Configure tracing, logging, and messaging options for the connector.

Each of these tasks is covered in more detail below.

When you have complete these tasks, your next steps will be to use System Manager and WSAD-IE to create service projects from the configured components, and deploy the service projects to WebSphere Application Server. Those tasks are described in Chapter 8, “Deploying to WebSphere Application Server,” on page 65.

Creating business object definitions

To create the business object definitions to be used by the connector, see the *Business Object Development Guide*.

Note: WebSphere Business Integration adapters used with WebSphere Application Server use only application-specific business objects, not generic business objects. All references to business objects throughout this book refer to application-specific business objects. Generic business objects are used in business integration systems based on the WebSphere InterChange Server integration broker. Some of the books in the WebSphere Business Integration adapters library, such as the *Business Object Development Guide* are also part of the WebSphere InterChange Server library and refer to both types of business objects.

The options for creating business object definitions are as follows:

- Use an ODA (Object Discovery Agent) to generate application-specific business object definitions. The ODA examines specified objects in the application, “discovers” the elements of those objects that correspond to business object attributes and their attributes, and generates business object definitions to represent the information. Business Object Designer provides a graphical interface to access the Object Discovery Agent and to work with it interactively. Refer to the adapter user guide for the connector you are configuring to determine whether an ODA (Object Discovery Agent) is provided.
- If no ODA is included with the business integration adapter, you can use the Object Discovery Agent Development Kit (ODK) to develop an ODA and then run it against the application.
- Create business object definitions manually using the Business Object Designer tool.

In addition, many adapters come with sample business objects. If they are included, the samples are located in the following product directory:

```
ProductDir\connectors\ConnName\Samples
```

To start Business Object Designer, use the Run command from the Start menu and browse for the BusObjDesigner.exe file. Alternatively, you can create and use a Windows shortcut to BusObjDesigner.exe

For more information about using Business Object Designer, see the *Business Object Development Guide*.

Once you create the business object definitions for application-specific business objects, read Appendix E, “Using Visual Test Connector,” on page 159, which explains how to test the business object definitions once you have created them.

Adding business objects to ICLs

For a connector to make use of a business object, the business object must be designated as a supported business object for that connector. You must designate the supported business objects when you configure the connector using Connector Configurator. To do that, you must first add the business object definition files to the same integration component library that contains the connector.

This is done through System Manager, as follows:

1. In the System Manager perspective, under Integration Component Libraries, expand the folder for an Integration Component Library project that you have created.
2. Select the folder *Business Objects*.
3. From the System Manager menu, choose File>Import.
4. The Import (Select) dialog appears. Select *File system* and choose Next.
5. The Import (File System) dialog appears. Use the Browse menu to navigate to a folder that contains business object definitions, in the form of *.xsd files, that you want to use.
6. Check the boxes for the files you want to use. By default, check *Create selected folders only*. Choose Finish.
7. After the dialog closes, select the Business Objects folder again in your Integration Component Library project folder. Right-click and choose Refresh View. The business object definitions that you imported will now be visible in your *Business Objects* folder.

Configuring WebSphere MQ queues for WebSphere Application Server

To enable WebSphere Application Server to work with a connector, you need to create and configure WebSphere MQ queues.

The WebSphere Application Server environment allows you to map the names of physical WebSphere MQ queues to the JNDI names used in the binding properties of WebSphere Application Server. If circumstances at a later time require you to use a different queue or queue manager, you can change that name in the local configuration file, and change it in the WebSphere Application Server Administrative Console, without having to regenerate .wsdl files.

Creating and configuring the queues includes the following tasks:

- Create WebSphere MQ queues that correspond to the interaction patterns you are implementing. This task is described in “Creating the WebSphere MQ queues” on page 54
- Specify the queue manager and queue names in the configuration properties for the connector. This task is described in “Saving configurations as ICLs” on page 65
- Export the connector configuration file into a WebSphere Application Server project and configure the queue names within the project files, so that the physical queue names acquired from the configuration file are associated with JNDI names that you specify in the Administrative Console of WebSphere Application Server. This task is described in “Deploying user projects” on page 67.
- In the Administrative Console of WebSphere Application Server, establish JNDI names that correspond to each of the queues. This task is described in Chapter 9.

“Message queues” on page 39 provides information about how WebSphere MQ queues are used in the WebSphere business integration system. “Setting the connection mode with the queue manager” on page 60 explains how to specify the connection mode in the connector’s configuration file.

Creating the WebSphere MQ queues

The business integration system requires that you create queues with the properties listed below.

Note: When you configure the connector, under “Specifying the queues to be used by the connector” on page 60, you will need to specify the name of each of these queues as a standard property in the connector’s configuration file.

- **DeliveryQueue:** Delivers event delivery messages from the connector framework to the integration broker.
- **RequestQueue:** Delivers Hub One Way or Hub Request request messages from the integration broker to the connector framework.
- **ResponseQueue:** Delivers response messages from the connector framework to the integration broker, in reply to Hub Requests.
- **FaultQueue:** Delivers fault messages from the connector framework to the integration broker. The connector framework places a message on this queue when it is unable to place the message on the reply-to queue.
- **SynchronousRequestQueue:** Delivers request messages from the connector framework to the integration broker.
- **SynchronousResponseQueue:** Delivers response messages from the integration broker to the connector framework, in reply to the requests in the SynchronousRequestQueue.
- **AdminInQueue:** Delivers administrative messages from the integration broker to the connector framework.
- **AdminOutQueue:** Delivers administrative messages from the connector framework to the integration broker.

Ways to define queues

You can configure the WebSphere MQ queues needed for your adapter, using any of the following methods:

- Customize and run a batch file provided with WebSphere Business Integration adapters.
- Use WebSphere MQ Explorer.
- Issue WebSphere MQ commands.

Tip

To make it easy to identify the connector with which a queue is associated, use the name of the connector as a prefix in the queue name. For example, name the Clarify connector’s event delivery queue: `clarifyconnector/deliveryqueue`.

Using WebSphere Business Integration adapter batch files to configure

WebSphere MQ queues: WebSphere Business Integration adapters provide a set of batch files that you can run to configure the WebSphere MQ queues needed for the adapters you are deploying. The batch files, located in *ProductDir\templates*, consist of:

- **configure_mq.bat** (Windows)
configure_mq (UNIX)
Run this batch file to configure the WebSphere MQ queues specified in `crossworlds_mq.tst`
- **crossworlds_mq.tst**
Edit this file to specify the WebSphere MQ queues in the business integration system. This file is read as input by `configure_mq.bat` and `clear_mq.bat`, a batch file provided with your WebSphere Business Integration adapter to clear messages from WebSphere MQ queues.

For more information about using `clear_mq.bat`, see “Clearing messages from WebSphere MQ queues” on page 108.

The contents of the `crossworlds_mq.tst` file are shown below. You can use this one file to specify the queues needed by each adapter you are configuring. Edit the file as follows:

1. Delete the statements:
`DEFINE QLOCAL(IC/SERVER_NAME/DestinationAdapter)`
`DEFINE QLOCAL(AP/DestinationAdapter/SERVER_NAME)` These apply only to business integration systems that use WebSphere InterChange Server.
2. For each adapter you are deploying, create a separate set of queue definition statements using as a template the statements beginning with `DEFINE QLOCAL(AdapterName/AdminInQueue)`.
3. If you are using bindings mode with remote queue definitions, customize the statement, `DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN) TRPTYPE(TCP)`, with the requested information for each queue manager you need to configure. If you are using client mode for your queue configuration, leave the statement as is. For more information about supported queue configurations, see “Defining the queue configuration” on page 56.

```

*****/
*   For each JMS queue (delivery Transport is JMS),
*   default values follow the convention:
*       AdapterName/QueueName
*****/
    DEFINE QLOCAL(AdapterName/AdminInQueue)
    DEFINE QLOCAL(AdapterName/AdminOutQueue)
    DEFINE QLOCAL(AdapterName/DeliveryQueue)
    DEFINE QLOCAL(AdapterName/RequestQueue)
    DEFINE QLOCAL(AdapterName/ResponseQueue)
    DEFINE QLOCAL(AdapterName/FaultQueue)
    DEFINE QLOCAL(AdapterName/SynchronousRequestQueue)
    DEFINE QLOCAL(AdapterName/SynchronousResponseQueue)
*****/
*   Define the default CrossWorlds channel type           */
*****/
    DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN) TRPTYPE(TCP)
*****/
*   End of CrossWorlds MQSeries Object Definitions       */
*****/

```

Using WebSphere MQ Explorer to configure WebSphere MQ queues: For information about configuring queues using WebSphere MQ Explorer, open WebSphere MQ Explorer and refer to its online help.

Using WebSphere MQ commands to configure WebSphere MQ queues: For information about configuring queues using WebSphere MQ commands, see *WebSphere MQ: System Administration Guide* and *WebSphere MQ: Script (MQSC) Command Reference*.

Defining the queue configuration

The WebSphere business integration system supports several queue managers and queue configurations. The connector can communicate with the queue manager in any of the following modes.

Bindings mode

With bindings mode, the integration broker and the connector can communicate directly with the queue manager, without using a TCP/IP connection. The integration broker and the connector need to be installed on the same machine so that they can use the same queue manager. This is the default mode.

Bindings mode with remote queue definitions

If WebSphere Application Server and the connector are installed on separate machines, with each machine running its own queue manager, the connector and the integration broker can still communicate with their respective queue managers using bindings mode. However, you need to specify remote queue definitions as explained in the example below.

Suppose brokerQM is the queue manager used by the integration broker and connQM is the queue manager used by the connector. To enable communication between the two queue managers, you need to set up the following channel definitions:

- For each queue that transmits messages from the connector to the broker, a remote queue definition must be created on connQM that points to a local queue on brokerQM. This requirement applies to the following queues:
 - DeliveryQueue
 - SynchronousRequestQueue
 - FaultQueue
 - AdminOutQueue
- For each queue that transmits messages from the integration broker to the connector, a remote queue definition must be created on brokerQM that points to a local queue on connQM. This requirement applies to the following queues:
 - RequestQueue
 - AdminInQueue
- For reply-to queues: when the integration broker sends a request message to the connector framework, it specifies in the message header the queue manager and queue to which the response is to be sent. This is also true for requests sent from the connector framework to the integration broker. You must perform certain administrative tasks to have response messages routed to the correct reply-to queue. These are described in *WebSphere MQ: Intercommunication*.

Client mode

If WebSphere Application Server and the connector must use TCP/IP to communicate with their respective queue managers, then they must use a client mode connection. Communication occurs through a client connection that uses TCP/IP as its underlying transport.

Enabling the application for use with the connector

To allow the connector's application-specific component to deliver business data to and from the application, you must establish a dedicated user account for the connector on the application. You will need to specify the user ID and password of this account when you create the configuration file for the connector.

For most connectors, the application must be configured to implement the event detection mechanism. Once the application has been configured, it can detect entity changes and write event records to the event store. The information is then picked up by the connector and processed. You should create triggers only for business objects and operations that are to be processed by a WebSphere MQ message flow. Otherwise, the message queues will fill up with messages that are never removed for processing.

For detailed information about these tasks and others necessary to enable the application to work with the connector, refer to the adapter user guide for the connector you are configuring.

Configuring the connector

The Connector Configurator tool provides a graphical user interface for configuring the connector. When you are finished specifying values for the connector's configuration properties, the Connector Configurator generates a configuration file for the connector.

The connector configuration that you create through Connector Configurator is used as the source for the .wsdl file used in generating a Service Project for the interaction pattern in WebSphere Application Server, and it is also used as the local configuration file that is installed at the same location where the connector agent (also referred to as the adapter agent) is installed. When you have completed the steps for creating a configuration file, you will be able to both save the configuration as an Integration Component Libraries project in the System Manager perspective (so that it can be used in generating a *.wsdl file) and save the configuration as a file that can accompany the connector agent in the location where the agent is installed.

Important: If the business integration adapter is running on UNIX, you must create the configuration file using Connector Configurator on Windows and then copy the file to your UNIX machine. When you create the configuration file, make sure that you observe UNIX path and file name conventions when setting properties.

Depending on the adapter product that you have purchased, you may already have a starting point for your connector configuration, either as a connector definition file in the directory *ProductDir\connectors\repository\ConnName*, or as a sample configuration file. You can open such files in Connector Configurator, and then revise them to create a complete configuration file for the connector. Note that while the connector definition file provides some starting values for the configuration file, a completed configuration file contains all the standard and application-specific properties for the connector, and specifies its supported business objects. Appendix B, "Standard configuration properties for connectors," on page 129, describes these properties in detail.

If no connector definition file or sample configuration file has been provided, you must create a new configuration file for that connector, using Connector Configurator.

The sections below discuss Connector Configurator settings that apply to every connector working with WebSphere Application Server.

Running Connector Configurator

Use Connector Configurator to create a new connector configuration, to revise an existing connector configuration that already has been saved to the System Manager integration component libraries, or to open an existing connector connector file that exists outside of System Manager.

Regardless of how you begin the process, in order to use the connector configuration with WebSphere Application Server as the integration broker, you must save the completed connector configuration to a project in the integration component libraries of System Manager.

To start and run Connector Configurator, do one of the following:

If you are creating a new connector configuration:

1. Open the System Manager perspective.
2. In the System Manager perspective, expand the Integration Component Libraries folder and select an existing Integration Component Library project folder. (If one does not already exist, you will need to create a new one. To create a new Integration Component Library project folder, select Integration Component Libraries, and right-click to open the New Integration Component Library dialog. Enter a project name for the new library project, and click Finish. The new library project appears as a folder within the top level Integration Component Libraries folder.) Expand the folder for the the Integration Component Library project that you are using, and select Connectors within it.
3. From the System Manager menu bar, click Tools>Connector Configurator. The Connector Configurator window opens and displays a New Connector dialog box.
4. Enter a name for the new connector configuration.
5. Click the pull-down menu next to System Connectivity: Integration Broker, and select WAS.
6. Select a template if one exists. If you do not have a template, choose None.
7. Before you make any settings in the configuration file, click the Standard Properties tab and verify the following settings:
 - BrokerType is WAS
 - DeliveryTransport is JMS
 - WireFormat is CwXML

To edit an existing configuration that has been previously saved in System Manager:

1. In the System Manager perspective, expand the Integration Component Libraries icon and highlight Connectors.
2. Select a connector configuration listed in the Connector folder, right-click on it, and choose Edit Definition. Connector Configurator opens and displays the configuration for the connector, with the integration broker type and file name at the top.
3. Before you make any settings in the configuration file, click the Standard Properties tab and verify the following settings:
 - BrokerType is WAS
 - DeliveryTransport is JMS
 - WireFormat is CwXML

To use an existing file to configure a connector, you must open the file in Connector Configurator, revise the configuration, and then save the configuration to the Integration Components Library in System Manager.

Follow these steps to open a *.txt, *.cfg, or *.in file from a directory:

1. In Connector Configurator, click File > Open > From File.
2. In the Open File Connector dialog, select one of the following file types to see the available files:
 - Configuration (*.cfg)
 - ICS Repository (*.in, *.out)
Choose this option if a repository file was used to configure the connector in an ICS environment. A repository file may include multiple connector definitions, all of which will display when you open the file.
 - All files (*.*)
Choose this option if a *.txt file was delivered in the adapter package for the connector, or if a definition file is available under another extension.
3. In the directory display, navigate to the appropriate connector definition file, select it, and click Open.
4. Before you make any settings in the configuration file, click the Standard Properties tab and verify the following settings:
 - BrokerType is WAS
 - DeliveryTransport is JMS
 - WireFormat is CwXML

After you have opened Connector Configurator as described above, you can set the properties (including the properties that specify queues) and designate supported business objects, as described in this section. When you have completed the configuration, choose Save To Project.

Setting Standard and Connector-Specific properties

Configuration of a connector requires that you set values for both standard properties and connector-specific properties. For connector-specific properties, see the documentation for the specific adapter that you are using. For standard properties, see Appendix B, “Standard configuration properties for connectors,” on page 129 later in this guide.

Note that the following property settings are mandatory for connectors that use WebSphere Application Server as the integration broker:

- BrokerType must be set to WAS
- DeliveryTransport must be set to JMS
- WireFormat must be set to CwXML

Designating supported business objects

In order to designate business objects as supported by the connector, the business objects must first have been saved to projects in an Integration Components Library folder, as described in “Adding business objects to ICLs” on page 53. The following instructions assume that you have already done that.

To designate existing supported business objects for the connector:

1. Open the connector configuration in Connector Configurator. Choose the Supported Business Objects tab.

2. The Business Object Name drop-down box will appear. Initially, if you have not yet designated any supported business objects for this connector configuration, the first visible field is blank. Click the drop-down arrow.
3. A scroll list displays, showing all the business objects that are available to be designated as supported in this connector configuration. To be available, a business object must have previously been saved to the subfolder *Business Objects* in the same Integration Components Library project that contains the *Connectors* subfolder for this connector configuration.
4. To designate one of the listed business objects as a supported business object for this connector configuration, click on it in the drop-down field. The business object name is added to the first field in the drop-down, and a new blank field is added. Repeat the process in the new blank field, continuing until you have added all the intended business object designations.
5. When finished, choose File>Save to project. This saves your connector configuration as an Integration Component Library project, and an icon for it is displayed in the System Manager Perspective.

Specifying the queues to be used by the connector

In “Creating the WebSphere MQ queues” on page 54, you defined a set of queues to be used by the connector to communicate with WebSphere Application Server. In the Connector Configurator, click the **Standard Properties** tab and assign these queues to the connector by setting values for the following standard properties:

- DeliveryQueue
- RequestQueue
- ResponseQueue
- FaultQueue
- SynchronousRequestQueue
- SynchronousResponseQueue
- SynchronousRequestTimeout
- AdminInQueue
- AdminOutQueue

Setting the connection mode with the queue manager

The default connection mode is bindings mode. Specify client mode as follows:

1. In the Connector Configurator, click the **Standard Properties** tab.
2. Assign to the standard property, `jms.MessageBrokerName`, the following value: `QueueMgrName:[Channel]:[HostName]:[PortNumber]`, where the variables represent the following:

QueueMgrName

The name of the queue manager.

Channel

The channel used by the client.

HostName

The name of the machine where the queue manager is to reside.

PortNumber

The port number to be used by the queue manager for listening.

For example:

```
jms.MessageBrokerName = WMQIB.Queue.Manager:CHANNEL1:RemoteMachine:1456
```

Configuring logging and tracing options

Logging is used to communicate system messages, component state changes, failures, and tracing information. The following files are generated:

Table 8. Connector logging and tracing files.

Default file name and path	Description
Temporary log file: <i>ProductDir\broker_name_connector_name_tmp.log.</i>	During startup, the connector generates a temporary log file. This file contains all messages that are logged during startup, including connector properties and business object definitions that are passed to the connector framework. and the file is written to the directory in which the product is installed.
Connector log file: UNIX: A connector logs messages to STDOUT by default, then those messages are rerouted to <i>ProductDir/logs/connector_manager_ConnName.log.</i> WINDOWS: A connector logs messages to STDOUT by default, but can be configured to send to a local destination log file in the <i>ProductDir</i> directory.	The connector's log file is used to store messages issued by the connector. It also contains messages regarding WebSphere MQ communication errors.
Connector message file: <i>ProductDir\connectors\messages\ConnName_LocaleName.txt</i>	This file contains the full text for each message issued by the connector. You can use this file to look up the text of message IDs you see in the log file. If the locale specified in the connector configuration file is not supported, the file <i>ConnName.txt</i> is used.
Trace file: Defaults to STDOUT for both UNIX and Windows.	Contains trace messages as specified by the selected trace level.

The logging system is always active and provides an accurate monitor of the connector.

To troubleshoot a problem, you can turn on tracing. Trace messages help you monitor actions taken in components of the business integration system. Trace levels define the amount of detail written to the trace file. The higher the trace level, the more detail you receive. Tracing differs from logging in the following ways:

- Logging always occurs, but tracing can be turned on and off as needed.
- Tracing contains more detailed information than logging about the state of components and the actions taken by them.
- Logging and tracing settings are persistent after reboots.

Tracing is off by default because it produces messages that are more detailed than you normally need.

For information about viewing logging and tracing messages using LogViewer, see "Using Log Viewer to view connector messages" on page 115.

Configuring connector logging

To configure connector logging options, click the **Trace/Log Files** tab and specify the following:

1. To have log messages routed to STDOUT, click the **To console (STDOUT)** check box.
2. To have log messages routed to a file, click the **To file** check box and specify the full-path name of the log file you want to use. You can have log messages routed to the console and to a file by specifying both the “To console” and “To file” options.
3. If you designated the use of a log file, also specify the following options:
 - a. To limit the size of the log file, set **Log file size** to a numeric value and unit of measure.
 - b. To permit the log file to grow with no limit, click the **Unlimited** check box.
 - c. If you have set a maximum size for the log file and you want to use file archiving, set **Number of archives** to the number of archive files you want to maintain.

For more information about managing log files, see “Managing log and trace files” on page 108.

Configuring connector tracing

To configure connector tracing options:

1. Click the **Trace/Log Files** tab.
2. To have trace messages routed to STDOUT, check **To console (STDOUT)**.
3. To have trace messages routed to a file, check **To file** and specify the full-path name of the trace file you want to use. You can have trace messages routed to the console and to a file by specifying both the **To console** and **To file** options.
4. If you designated the use of a trace file, also specify the following options:
 - a. To limit the size of the trace file, set **Trace file size** to a numeric value and unit of measure.
 - b. To permit the trace file to grow with no limit, check **Unlimited**.
 - c. If you have set a maximum size for the trace file and you want to use file archiving, set **Number of archives** to the number of archive files you want to maintain.
5. To set the tracing level:
 - a. Click the **Standard Properties** tab.
 - b. Set the **AgentTraceLevel** property to one of the values listed in Table 9 on page 63.

Refer to the adapter user guide for the connector you are configuring for more details about the information generated by the different trace levels for that connector. You can set connector tracing to one of the following levels:

Table 9. Connector trace levels.

Trace level	Description
1	Traces initialization and the sending and receipt of business objects.
2	Prints messages for level 1. In addition, provides more details than Level 1 for the same types of events.
3	Prints messages for levels 1 and 2. In addition, traces the exchange of messages between the connector agent and the messaging driver.
4	Prints messages for levels 1 through 3. In addition, traces the passing of business objects between internal levels of the connector.
5	Prints messages for levels 1 through 4. In addition, traces the passing of administrative messages between internal levels of the connector.

A new or changed tracing level takes effect when you restart the connector.

For more information about archiving trace files, see “Managing log and trace files” on page 108.

Configuring the connector startup files, shortcuts, and environment variables

The procedure for starting a connector and the set-up tasks required both depend on the platform where the connector is running.

For Windows

When you install WebSphere Business Integration adapters on Windows, a shortcut is created for each installed connector on the WebSphere Business Integration Adapters program menu (Start > Programs > IBM WebSphere Business Integration Adapters > Adapters > Connectors). You must use the startup options listed in Appendix C, “Connector startup options,” on page 153 to customize the following:

- The connector’s shortcut properties
- The connector’s startup file, `start_connName.bat` (for connectors written in Java)
- The connector’s startup file, `start_connector.bat` file (for connectors written in C++).

For UNIX

In the UNIX environment, you start a connector by running `connector_manager_connName` script, which is a wrapper for the generic connector manager script (`ProductDir/bin/connector_manager`). This wrapper includes the following information:

- The name of the connector to start or stop
- Appropriate command-line options of the generic connector manager. For example:
 - The SAP connector requires the `-t` command-line option. Therefore, its startup script already includes the `-t` option.
 - All UNIX connectors run with the `-b` option. Therefore, all connector startup scripts already include the `-b` option. To have a connector run in the foreground, remove the `-b` option from the generic connector manager script (`connector_manager`).
- The name of the configuration file.

If you have created a custom adapter or if you have installed an adapter using electronic software delivery (ESD), you need to do the following before you start up the connector for the first time:

1. Run the Connector Script Generator tool to update the `connector_manager_connName` script with the name of the connector's configuration file. See Appendix D, "Using the Connector Script Generator tool," on page 157, for more information about running this tool.

Alternatively, you can navigate to the `ProductDir/bin` directory and edit the `connector_manager_connName` file to specify the name of the connector's configuration file. In the file, locate the `AGENTCONFIG_FILE` variable and set it to the full-path name of the configuration file as follows:

```
AGENTCONFIG_FILE=ConfigFile
```

2. If desired, update your `PATH` environment variable to include the `ProductDir/bin` directory.
3. Ensure that the `CWSharedEnv.sh` file is sourced from the shell startup script (such as `.cshrc`) for your account.

Customizing the startup script: The generic connector manager script calls the appropriate `start_connector.sh` script, which is the actual script that manages the particular connector. Each WebSphere Business Integration adapter includes a `start_connector.sh` script. You can modify the `start_connector.sh` script to include any of the supported startup options listed in Appendix C, "Connector startup options," on page 153.

Note: For information about creating a startup files for connectors, see *Connector Development Guide for C++* or *Connector Development Guide for Java*.

Using Visual Test Connector to verify your interfaces

Visual Test Connector simulates the activities of a connector to allow you to test your integration components without the complexity of running an actual connector. You can use Visual Test Connector to verify that you have configured your source and destination connectors correctly and that you have properly specified their supported business object definitions. See Appendix E, "Using Visual Test Connector," on page 159 for detailed information on how to use Visual Test Connector.

Chapter 8. Deploying to WebSphere Application Server

Before you perform the tasks in this chapter, you should have done the following:

- Created and configured JMS queues for the data transactions you wish to implement
- Configured your connectors with their associated business objects, using Connector Configurator.

This chapter describes how to

1. Save connector configurations and associated business objects as Integration Component Libraries in System Manager.
2. Create User Projects, consisting of groups of Integration Component Libraries, in the System Manager perspective of WebSphere Studio Application Developer Integration Edition
3. From the System Manager perspective of WebSphere Studio Application Developer Integration Edition, deploy the System Manager user projects to the Business Integration perspective.
4. Use the Business Integration perspective to create the WebSphere Application Server components used for each interaction pattern, and use the WebSphere Application Server Administrative Console to complete the setup of the queues.
5. For each interaction pattern, create an Enterprise Application Archive project that can be deployed on a server.

Saving configurations as ICLs

The WebSphere Business Integration adapters use configured resources--referred to as integration components--that are managed through the System Manager perspective. Integration components are organized in sets, referred to as Integration Component Libraries (ICLs). The integration components commonly used for all adapters are connectors and business object definitions.

To implement an adapter, you must configure the connector and designate the business object definitions that the connector supports. You must save both the business object definitions and the connector configurations into Integration Component Library projects in System Manager. To do so:

1. In the System Manager perspective, right-click on the Integration Component Libraries perspective, and choose New Integration Component Library.
2. In the New Integration Component Library dialog, provide a name for the library in the *Project name* field, and choose Finish. A new folder, with the name you have chosen, is added to the Integration Component Libraries. The folder contains subfolders for several types of integration components; for the purpose of implementing a connector with WebSphere Application Server as the broker, you will require only the Business Objects and Connectors folders.
3. Select the Business Objects folder and choose File>Import. The Import dialog displays.
4. Select File System and choose Next. The Import File system dialog displays. In the Directory drop down list, browse to a folder where you have placed business object definition (.xsd) files that you want to use with the connector in the interaction pattern that you have created.

5. Select the folder and choose OK. The Import dialog appears, with check boxes next to each of the available resources. Check the box for each .xsd file that you expect to use, and choose Finish.
6. In System Manager, in the Business Objects folder under Integration Component Libraries, right click on the Business Objects folder and refresh the view. Icons for the imported business object definitions will appear under the folder.
7. Select the Connectors folder, then, from the Tools menu, choose Connector Configurator. The Connector Configurator tool is activated, and the New Connector dialog displays.
8. You will use the Connector Configurator tool to set configuration properties and specify supported business objects for your connector. If you already have a sample or partial connector configuration file, cancel the New Connector dialog. Then, in the Connector Configurator dialog, choose File>Open>From file. The Open File Connector dialog appears. Navigate to your existing sample or partial connector configuration file, select it, and choose Open. The Connector Configurator properties screen, named for the selected connector configuration, displays.
9. If you are creating a completely new connector configuration, select WAS as the Integration Broker, provide a name for the new connector configuration, select a property template (choose None if you don't already have one), and click OK. The Connector Configurator properties screen, named for the new connector configuration, displays.
10. For any connector used with WebSphere Application Server, set configuration values in the following tab:
 - a. In the Standard Properties tab, set required values for the use of your integration broker, including the queues that are used to implement the interaction patterns. See "Specifying the queues to be used by the connector" on page 60 and "Configuring WebSphere MQ queues for WebSphere Application Server" on page 53.
 - b. In the Supported Business Objects tab, specify the business objects supported for use with the connector you are implementing. See "Designating supported business objects" on page 59
 - c. When you have completed the tabs, choose File>Save as>To Project. The Save As dialog appears. Choose the Integration Component Library into which you want to save this connector configuration, enter the name you want to use for the connector, and choose Save. The Connector Configurator validates your connector configuration and displays a message for either success or failure. If you have no more changes to make to the configuration properties of this connector, close the Connector Configurator.
11. In the System Manager perspective, under the Integration Component Library project that you created, an icon for your connector configuration is now displayed under the Connectors folder.
12. Your connector is now configured as part of an Integration Components Library, with the properties and supported business objects that you specified. If you need to make a change in the configuration, you can do so by double-clicking this specific connector icon to open its configuration file in Connector Configurator. Note that if you wish to add additional supported business objects to the configuration for this connector, you must first save the business objects into the same Integration Component Library in which this connector resides. Then you can open Connector Configurator, select the

Supported Business Objects tab, and click in the Business Object Name drop down box to see the additional available business objects.

Creating user projects in System Manager

System Manager user projects specify the integration components that will be used for some particular purpose, such as a specific interaction pattern. After you have created System Manager user projects and populated them with integration components, you can deploy the user projects to the Business Integration perspective, for use in the creation of WebSphere Application Server Enterprise Application Archive projects.

User projects are created within the System Manager perspective, using the New User Project dialog. To create a user project, open the System Manager perspective, and do the following:

1. Under the User Projects folder, select and right-click on WAS Projects.
2. In the pop-up menu, select New User Project. The New User Project dialog appears. In the *Project name* field, enter a name that you want to use for the user project.

If the dialog contains a prompt to *Select the type of User Project to create*, choose *WebSphere Application Server Project*. (Depending on the configuration of your installation of WebSphere Business Integration adapters, *WebSphere Application Server Project* may be the only choice available.) The Available Integration Component Libraries field shows folders for any integration component libraries that you have created, and includes check boxes for both the libraries and the individual integration components that they contain. Expand the folders and check the boxes for all the components that you want to make available in the user project that you are creating. Choose Finish. (If you have not yet created any integration component libraries, none will be displayed. However, you can still create the user project at this time. At a later time you can right-click the user project that you have created and choose *Update project* to add any new integration component libraries that you have created.)

Note: You can also access the New User Project dialog through the Business Integration perspective by choosing New>Project>WebSphere Business Integration>User Project. When the new user project is created, the System Manager perspective is opened automatically; the new user project is created in the System Manager perspective. That does not change the other procedures for this task.

3. The User Project is created and appears under the WAS Projects folder in the System Manager perspective.

Deploying user projects

After you have created a user project and added integration component libraries to it, you can deploy that project to generate files that can be used in the WebSphere Application Development Environment, Integration Edition for the creation of WebSphere Application Server projects. To do so:

1. In the System Manager perspective, right-click on the icon for your user project (make sure that you are within the User Projects folder, not within the Integration Component Libraries folder). A pop-up menu appears.
2. Choose Deploy WAS Project. The Component Selection Page appears, listing the specific integration components available in the user project you have selected. (You should see a connector and one or more business objects. If you do not

see business objects, verify that you designated business objects as supported when you configured the connector, that you included that connector configuration and those business objects in an integration component library, and that you updated the user project with that integration component library.) Check the boxes for the components that you want to include, and click Next. A pop-up menu appears, providing three choices:

- Export to a Service Project

This option saves the connector configuration files, in WSDL format, and the business object definition files, in XSD format, directly into a WebSphere Studio Application Developer Integration Edition service project. The service project must already exist, and must reside within the current workspace.

- Export as a Jar File

This option saves the connector configuration files, in WSDL format, and the business object definition files, in XSD format, into a JAR file. The JAR file can be used locally or moved to another machine and installation of WebSphere Studio Application Developer Integration Edition to be imported into a Services Project.

- Export to a Directory

This choice saves the connector configuration files and business object definition files of your user project to the directory location that you select. The connector configuration files (file in XML format with a .cfg or .con extension) are transformed into Web Services Description Language (WSDL) format and renamed with the .wsdl extension. The business object definition files are saved in .xsd format.

To use these files in WebSphere Studio Application Developer Integration Edition, you will import them into a Service Project, as described in “Import files for a new service project” on page 69.

3. Select a choice and click Finish.

Files will be generated in the location that you specified. The files are an interface .wsdl file, a JMS bindings .wsdl file, and a JMS services .wsdl file, all specific to this connector configuration, and .xsd files corresponding to each of the business object definitions.

Creating WebSphere Application Server applications

In this task, you create WebSphere Application Server applications that implement interaction patterns. The applications include the following types of projects:

- Service Project

Contains the initial .wsdl and XSD files that were deployed from System Manager, generated .wsdl files with EJB bindings, and may contain generated helper classes.

- Enterprise Application Archive project

Contains one or more EJB projects (which contain generated Session and Message Driver Beans (MDBs))

In addition to WebSphere Application Server components (EJBs and MDBs), the tooling generates the following helper classes:

- Data-type java beans, generated for each XML Schema Definition found in the wsdl or xsd file, with format depending only on the schema.
- Format-handler java beans, used to serialize and deserialize data-type beans to and from wire format. The wire format (and therefore the format handlers) are

specific to the encoding and binding, as specified in the .wsdl file. For this solution, binding is jms and encoding is XML (reflected in the package name of the format handler class).

After you have created the projects, you can export an individual interaction pattern as an Enterprise Application Archive application for deployment on a WebSphere Application Server, or you can include multiple interaction patterns in a single Enterprise Application Archive application for deployment.

Note: The final Enterprise Application Archive project must include a helper file, `boutils.jar`, that contains the business object initialization library routine. You must add this file to the completed Enterprise Application Archive by importing it manually, as described under Adding the Business Object Library Routine, later in this section.

The topics in this section describe how to create for each interaction pattern.

Agent Delivery (Asynchronous Event Delivery)

The Agent Delivery interaction pattern asynchronously delivers a message from the adapter to WebSphere Application Server. No response message is required from WebSphere Application Server.

To create an Agent Delivery interaction pattern, you must create an Enterprise Application Archive application that contains:

- A message driven bean (MDB) that receives messages and invokes the EJB
- An EJB that is triggered by the MDB and that implements the business logic that you want to use for processing the event
- Helper classes that represent the .xsd files as java beans, and format handler java beans that convert XML to java, as specified in the .wsdl file.

The Enterprise Application Archive application must also contain .wsdl and .xsd files from the Service Project. If you chose Export to Service Project when you deployed your System Manager user project, the Service Project already exists and is included in the Enterprise Application Archive. If you instead exported your System Manager user project as individual files to a directory or as a JAR file, you will need to import the file or files into a new Service Project.

Import files for a new service project

This step is not necessary if you exported your System Manager user project directly into an existing service project, as described in “Deploying user projects” on page 67. If you exported from System Manager to a JAR file or to a directory, you will need to perform this step to import the files into a service project. This step creates a Session Bean and .wsdl files that describe it.

This procedure assumes that in System Manager you chose the option to Export to a Directory. To import the exported files into a new service project, do the following:

1. In the Business Integration perspective, choose File>New>Service Project.
2. The New Service Project dialog displays (see Figure 19). Enter a name that you want to use for this project, accept the default for Project contents, and click Next.

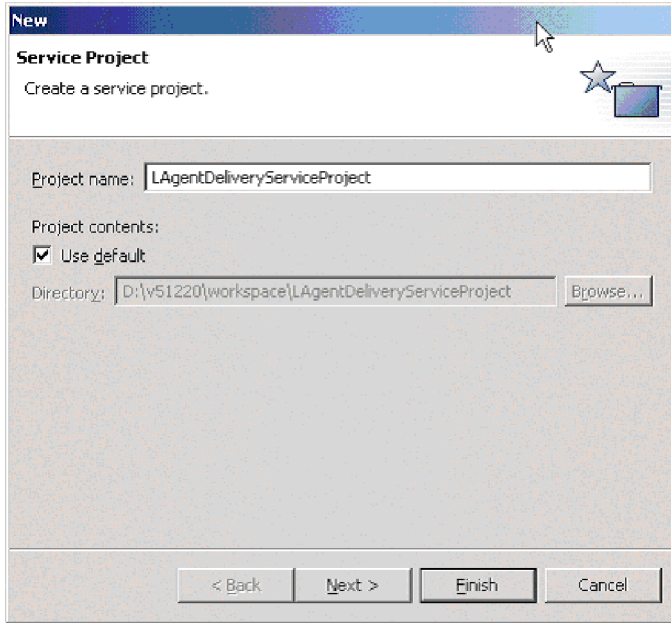


Figure 19. New Service Project dialog

3. Click Finish. The new service project is created in your default workspace. The core libraries, and any additional elements that you included from existing projects, are displayed in the Package Explorer view of the Business Integration perspective.
4. In the Package Explorer view, right-click on the new service project you have created, and choose Import.
5. The Import dialog (see Figure 20) displays. Select the File System folder and click Next.

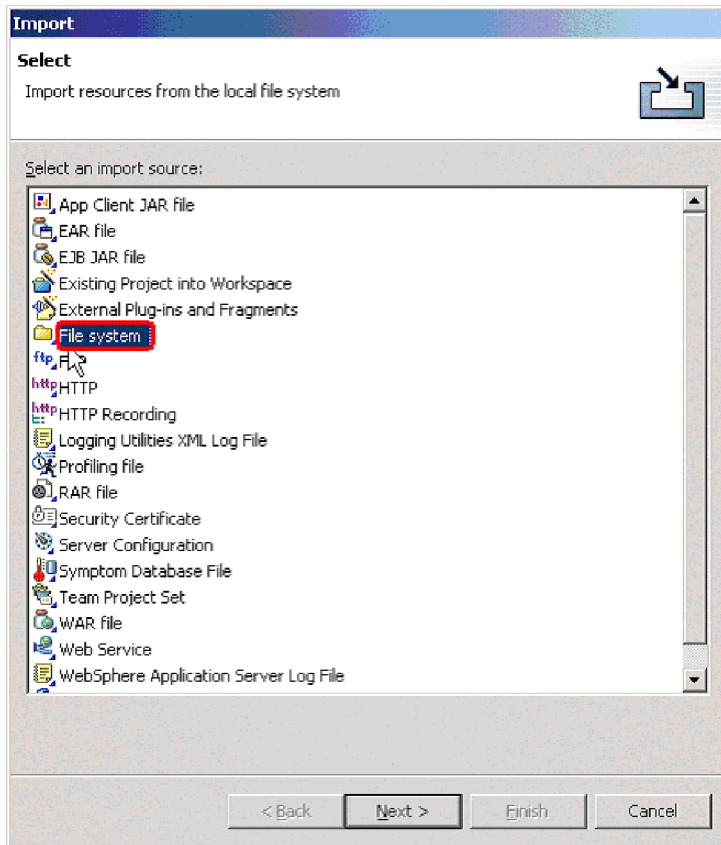


Figure 20. Import dialog

6. In the Directory scroll box, browse to the folder that contains the .wsdl files and .xsd files that you have created for use in this interaction pattern. If you created the .xsd files and .wsdl files for this interaction by deploying a user project from System Manager, you should have a folder with the same name that you gave the user project. Choose that folder, and click OK.
7. The Import dialog now displays the files contained within the project folder that you have selected (see Figure 21). Checkboxes enable you to include or exclude specific .xsd or .wsdl files. Your default choice, assuming that the user project that you created in System Manager contains only elements that are usable in the interaction pattern that you are creating, should be to include all the .xsd and .wsdl files displayed in the folder. However, the dialog enables you to select any or all of the files. Typically you should have all of the business object .xsd files that you designated as supported in the connector configuration in the same folder with the WSDL file that was created from the connector's configuration file. Error messages may be displayed if the .xsd files for any supported business objects are missing from the folder that contains the WSDL file.

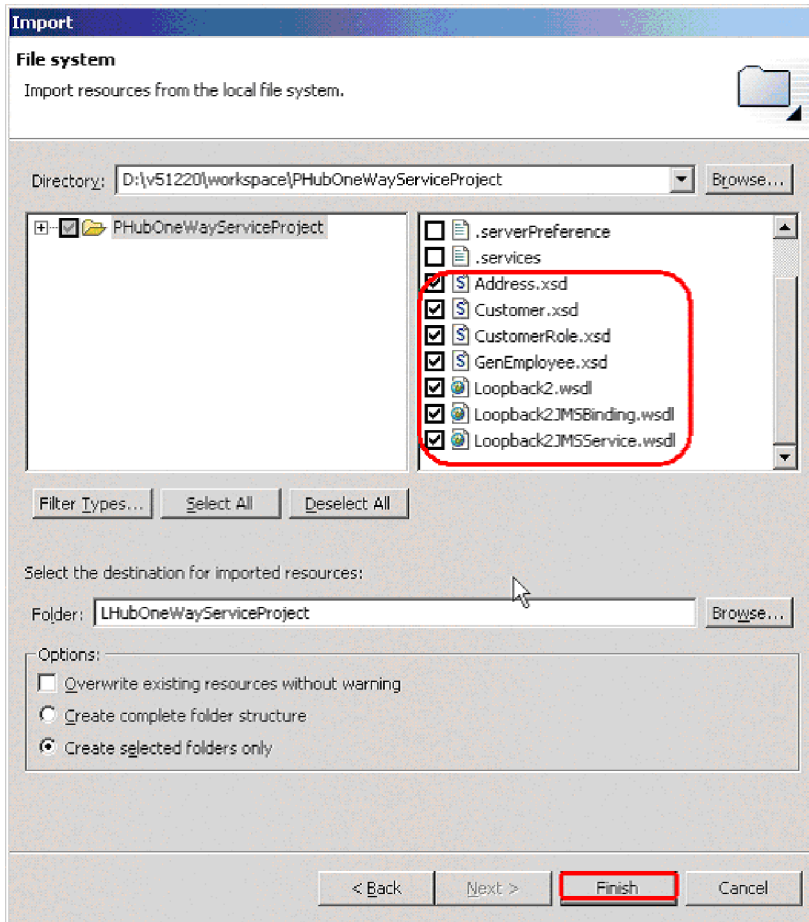


Figure 21. Selected Project folder files

8. Specify the name of the destination folder for the files that you are importing; for example, MyServices\com\ibm\cw
9. Click Finish.
10. In Package Explorer, the files that you imported are displayed under the service project that you named.

Create an EJB

The Agent Delivery interaction pattern requires an EJB to consume the message incoming to WebSphere Application Server from the agent. You must both create the EJB and add business logic to the EJB method to process the received event.

Start by creating a project that will contain the EJB:

1. In the J2EE view of the Business Integration perspective, choose File>New>Project.
2. In the New Project dialog, choose EJB in the left-hand panel, then select EJB project in the right-hand panel, and then choose Next (see Figure 22).

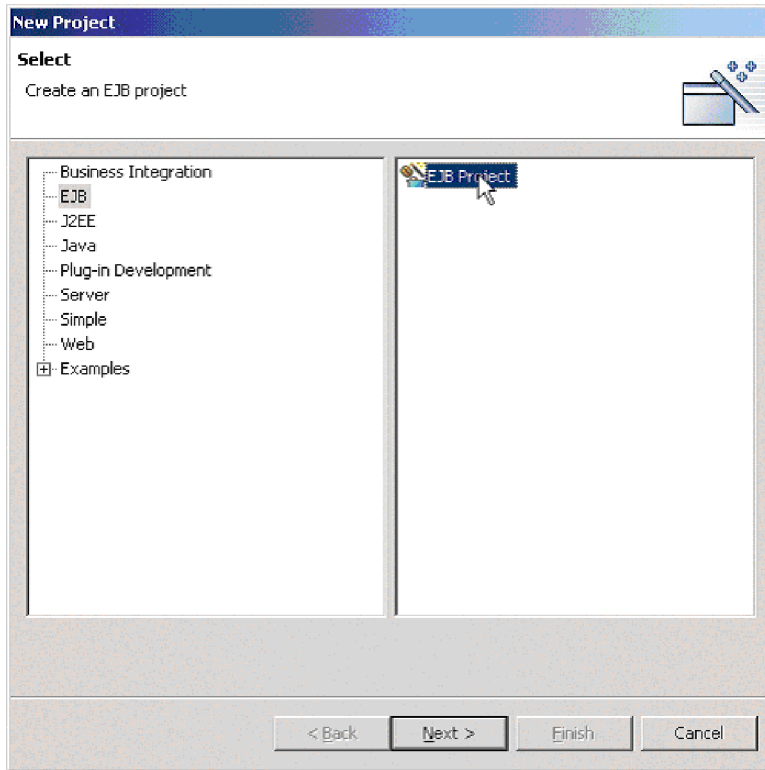


Figure 22. Choosing EJB in the New Project dialog

3. In the Select an EJB Version dialog (see Figure 23 on page 74), choose Create 2.0 EJB Project, and click Next.

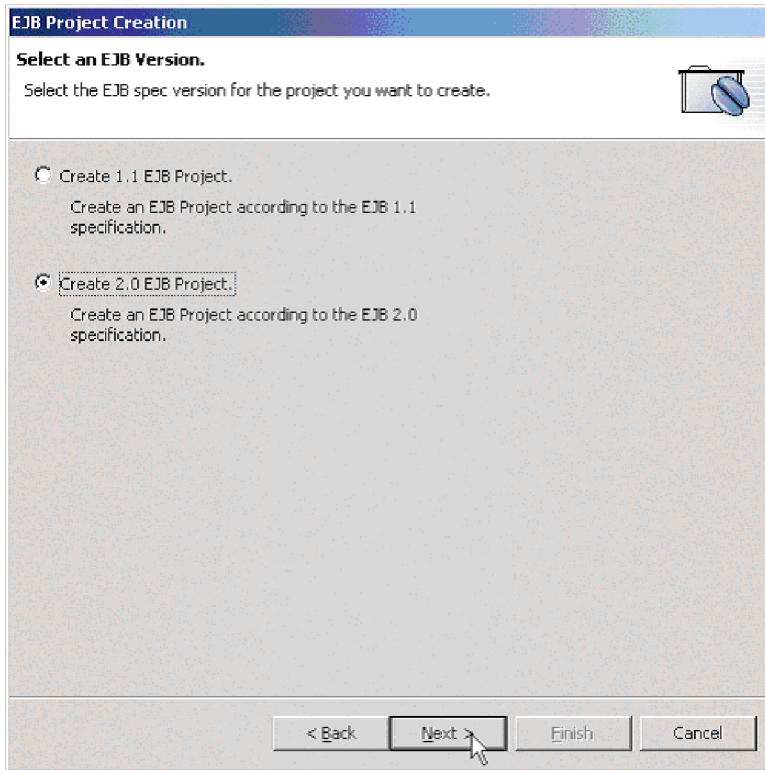


Figure 23. Select an EJB Version dialog

4. In the EJB Project Creation Dialog (Figure 24), provide a name for your EJB project, and the name of a new or existing Enterprise Application Archive project folder to which it will be added. If you specify a name for a new Enterprise Application Archive project, a folder with that project name will be created.

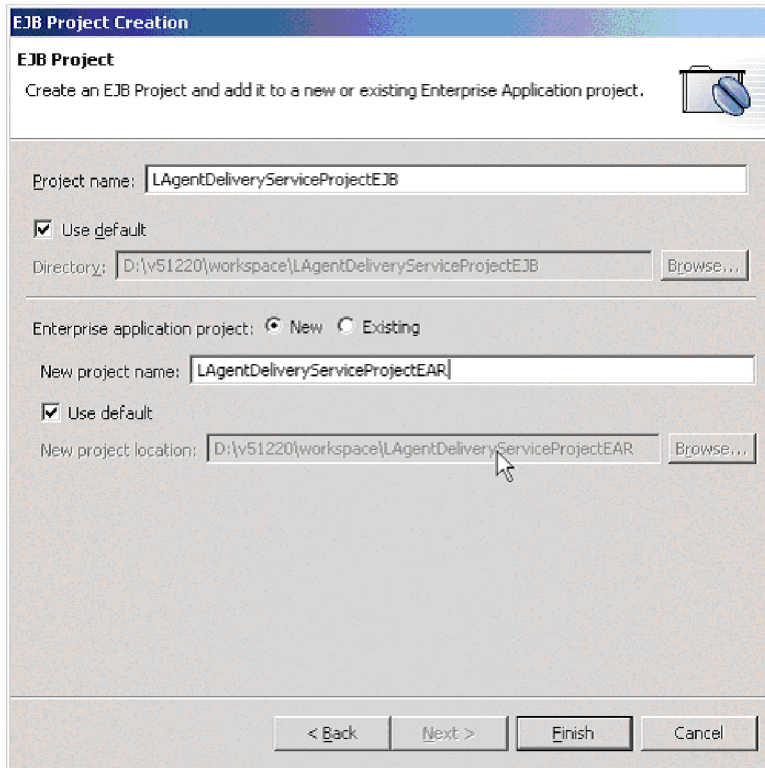


Figure 24. EJB Project Creation Dialog

5. Choose Finish. A project, containing an EJB module, is generated under the name that you specified. The EJB project is created for you and you can see it in the J2EE Hierarchy view under EJB Modules. In the Services view, under Service Projects, select the .wsdl interface file that you imported from your connector configuration (for example, AgentDelConnector.wsdl). Right click, and choose New>Build from Service. The New Service Skeleton dialog opens.
6. Choose EJB Service Skeleton (see Figure 25) and click Next.

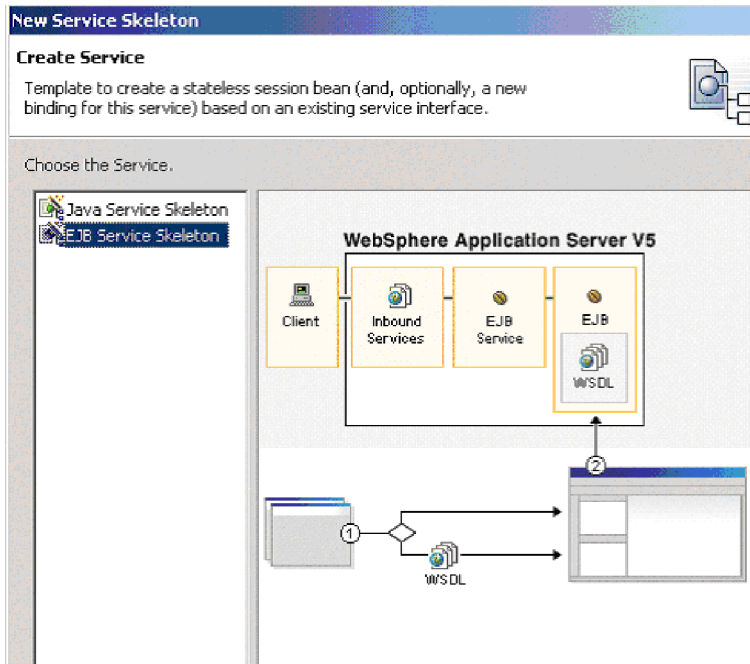


Figure 25. EJB Service Skeleton

7. In the New Service Skeleton dialog, select *Create a new port and binding* and select *Generate helper classes*. Choose Next.
8. The Service Skeleton dialog appears (see Figure 26).

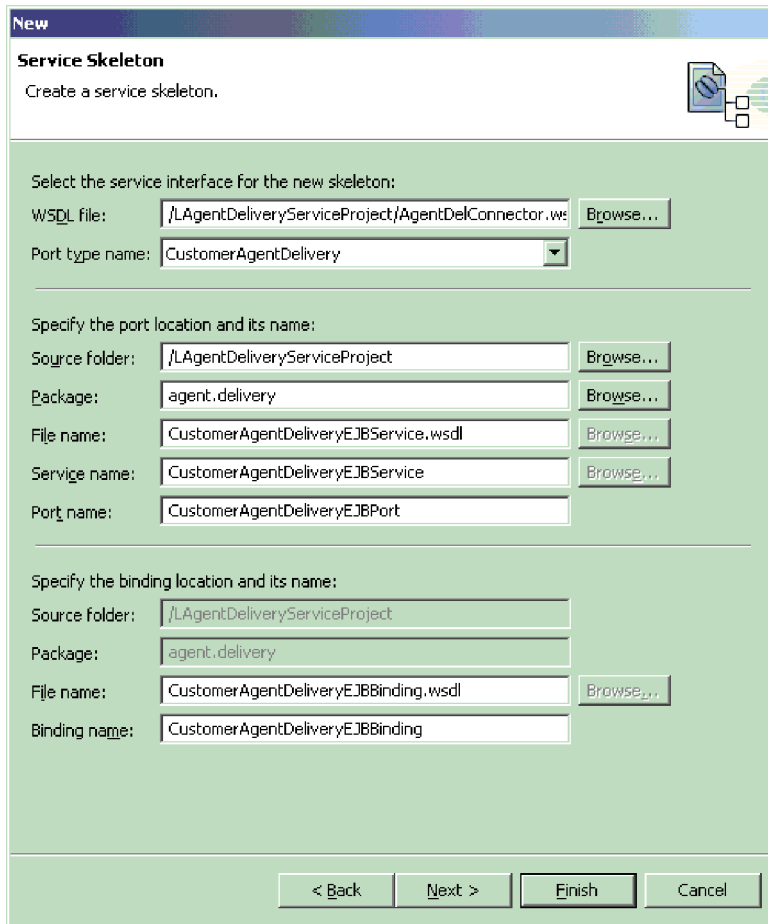


Figure 26. Service Skeleton dialog

9. Select the appropriate Port type name according to the interaction pattern you are creating, and provide a meaningful package name in place of the default. Click Next.
10. In the next dialog, you can specify properties for generating the EJB skeleton, or accept the defaults. Click Finish. The newly created EJB project will be listed under Deployable Services in the Services view.
11. Edit the skeleton to add any necessary business logic.

Create an MDB

A message driver bean is responsible for receiving events as messages from the connector and invoking EJB processing of the events.

In this step, you will deploy the EJB service to the server using existing inbound port and bindings - the original JMS bindings. This step generates an MDB that invokes the appropriate method of the Session Bean skeleton. The MDB is specific to the port type and the selection of the operation is made based on the incoming message properties: WSDLOperation (and possibly WSDLInput and WSDLOutput) properties. The additional helper classes, format handlers, are also generated during the deployment. These are used to convert from the wire format (the XML message) to the Java Class, which in turn is passed as an argument to the business method of the EJB.

You must configure the JNDI names used by the EJB and MDB. The JNDI setup is necessary since the lookup names of the queue connection factory and queues defined in WSDL may not match the actual names defined on your server. The WebSphere Application Server allows you to map the name you are looking up (as specified in WSDL) to the actual JNDI name deployed on the server, thus allowing correct execution without the need for modification of your wsdl files. You also need to specify what listener port is used by the MDB.

1. In the Package Explorer view, expand the folder for the Service Project that you have created and select the EJBService .wsdl file that was generated in the previous task. Right-click on the file, and in the pop-up choose Enterprise Services>Generate Deploy Code (see Figure 27).

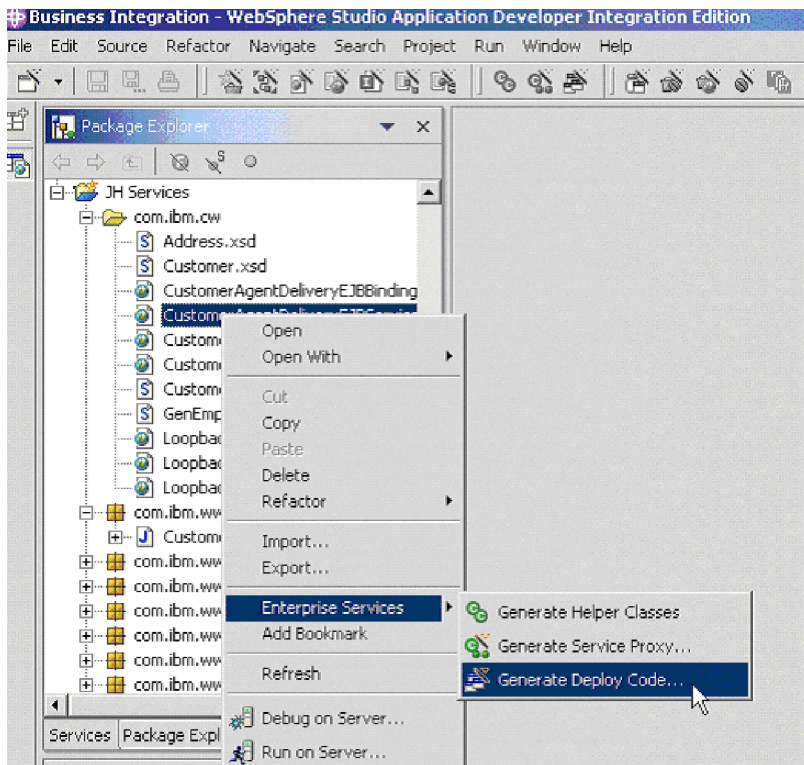


Figure 27. Choosing Generate Deploy Code

2. In the Generate Deploy Code dialog, select *Use an existing port*, and choose Next (see Figure 28).

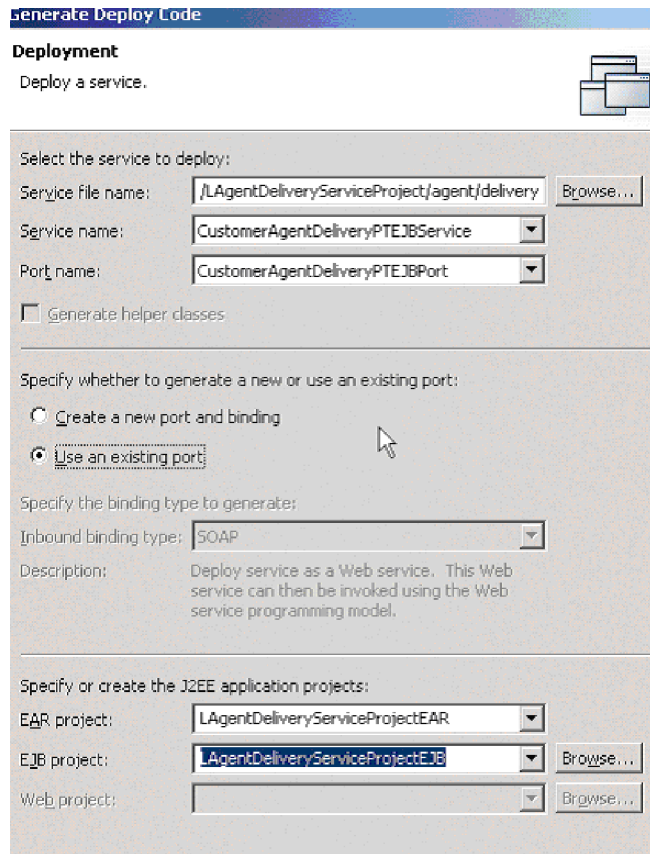


Figure 28. Generate Deploy Code dialog

3. In the Inbound Service Files dialog, select the JMSService .wsdl file for your project, and the corresponding service name and port name. Choose Finish.
4. In the J2EE Hierarchy view, select your project under EJB Modules, right-click, choose *Open With the EJB Deployment Descriptor* editor and choose the Bean tab (see Figure 29). Verify that the Destination type is Queue. Specify the correct ListenerPort, so that its queue connection factory and queue match those specified in the .wsdl file for this interaction pattern.

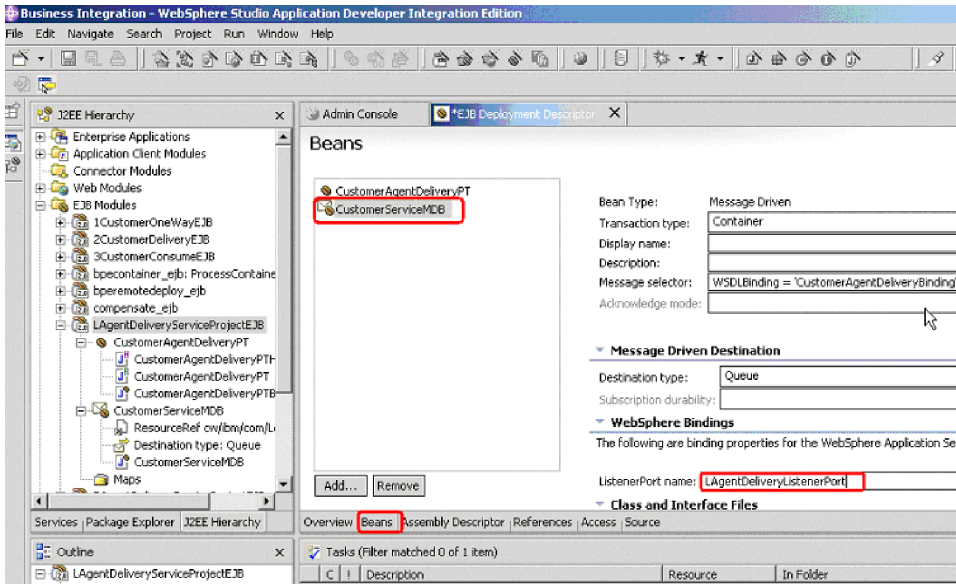


Figure 29. Choosing the Bean tab

5. Choose the References tab (see Figure 30). Select the resource queue name listed under the MDB. The name from the .wsdl file then appears in the *Name* box at right. Under WebSphere Bindings, in the *JNDI name* field, enter the correct JNDI name for the deployment of this queue, as defined in the WebSphere Application Server Administrative Console.

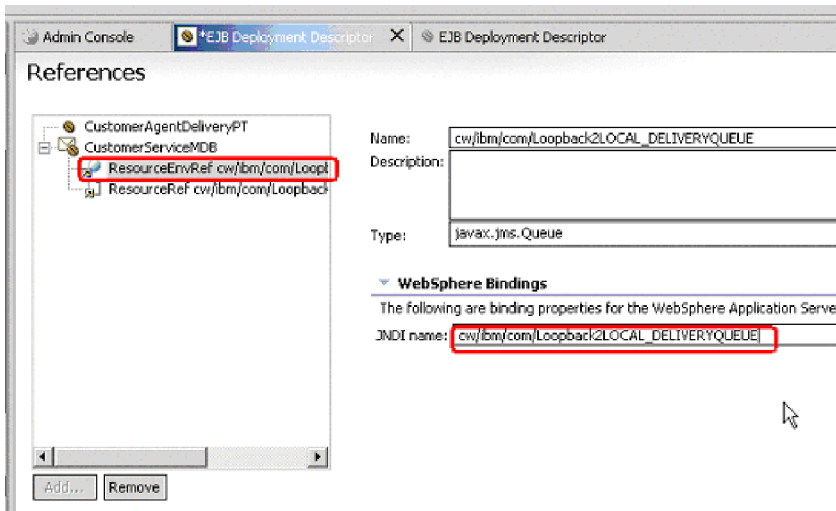


Figure 30. References tab view

6. Select the connector factory listed under the MDB. Under WebSphere Bindings, in the *JNDI name* field, enter the correct JNDI name for the deployment of this connection factory, as defined in the WebSphere Application Server Administrative Console (see Figure 31).

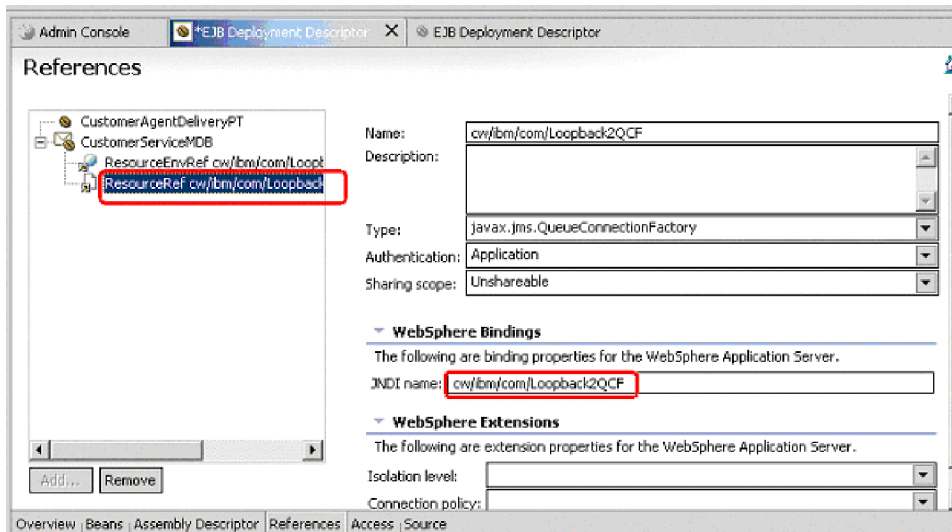


Figure 31. Entering the JNDI name

7. Save the changes that you made in the EJB Deployment Descriptor editor.
8. Open the Server Configuration tab in the lower panel of the Business Integration perspective and right-click on the Servers icon.
9. Choose New, and choose Servers and Server Configuration. The *Create a New Server and Server Configuration* dialog appears.

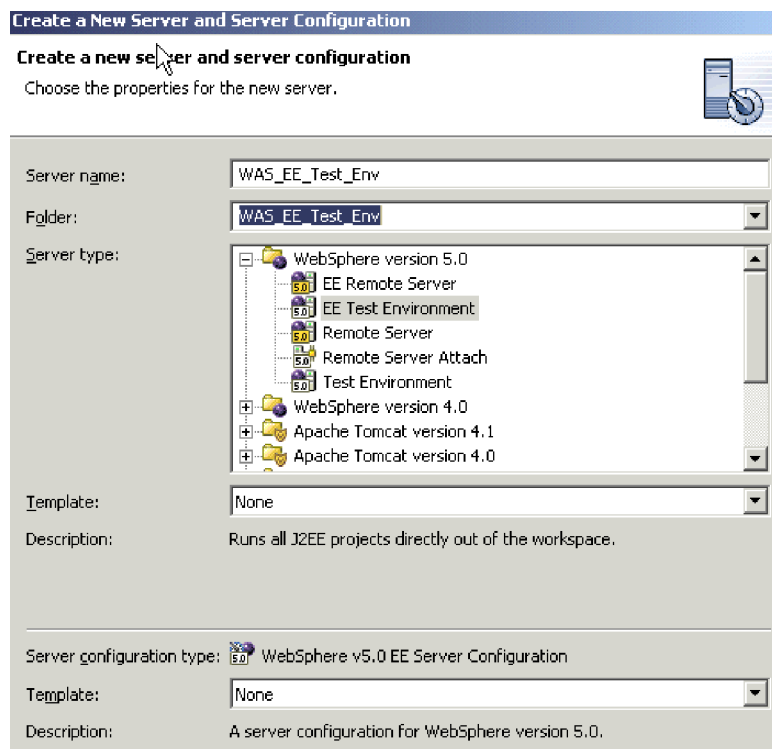


Figure 32. Create a New Server and Server Configuration dialog

10. In the Server name field, enter a name for the server instance you are creating. Enter the same name in the Folder field. In the Server Type field, select EE Test Environment.

11. Choose Finish, and respond Yes when prompted to create a new server project with the name you specified.
12. After the utility completes, in the Server Configuration panel double-click the icon for the new server instance that you have created. The WebSphere Server panel appears, with the name of the server instance you have created. Choose the Configuration tab.
13. The WebSphere Server Configuration panel appears (see Figure 33). Put a check in the Enable administration console check boxes.

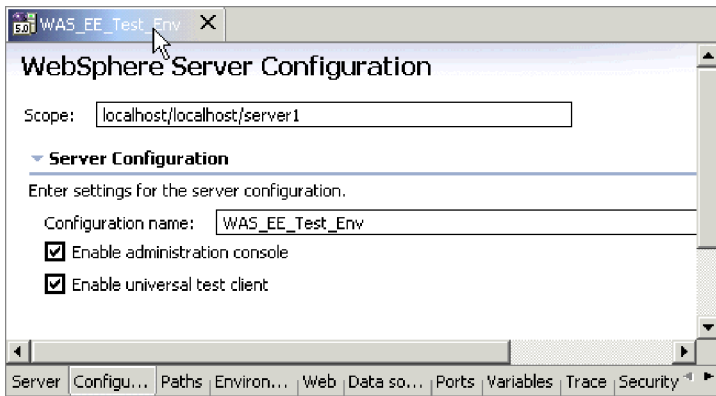


Figure 33. WebSphere Server Configuration panel

14. Choose the Variables tab. In the Defined variables drop down box, locate the MQ_INSTALL_ROOT variable. Change the value of the variable to the absolute root path of your MQ installation.
15. Save your configuration. (Ctrl+S).
16. Start the newly configured server instances, as follows: In the lower portion of the Business Integration perspective, choose the Servers tab. The Servers panel displays. In the Servers panel, select the icon for the newly configured server instance, and click the run button to start the server.
17. Open the WebSphere Application Server Administrative Console. Under General Properties under the Configuration tab, add the listener port name that you specified in the Bean tab of EJB Deployment Descriptor (see Figure 34). Add the same connection factory JNDI value and destination JNDI queue name value that you specified in the References tab of EJB Deployment Descriptor. As appropriate for your needs, make any necessary changes in the Maximum sessions, Maximum retries, and Maximum messages fields.

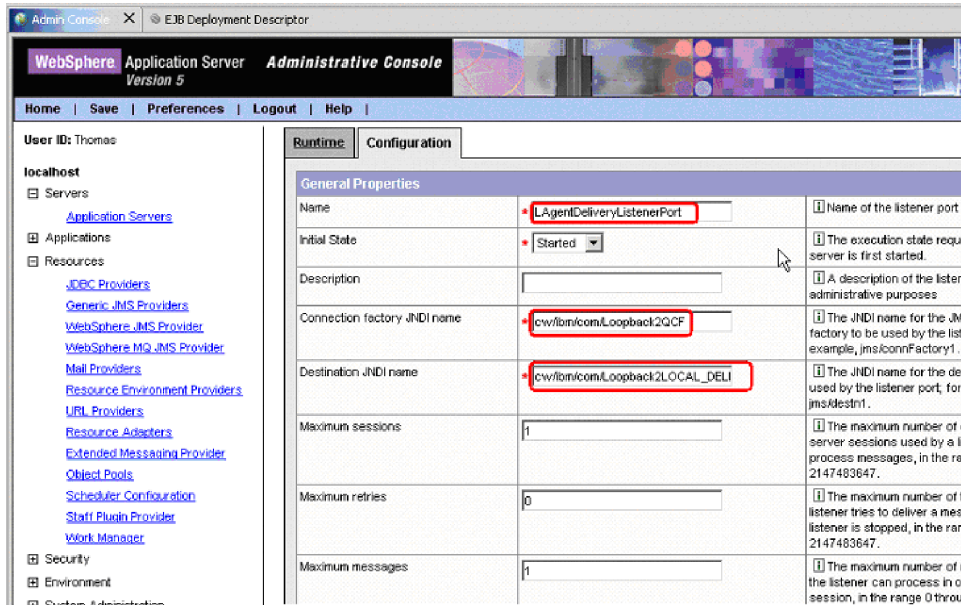


Figure 34. WebSphere Application Server Administrative Console, Configuration tab

- In the WebSphere Application Server Administrative Console, add the queue as a resource. Provide a display name for the queue and a JNDI lookup name (see Figure 35). The JNDI lookup name must be the same as specified for the EJB Deployment Descriptor and the listener port.

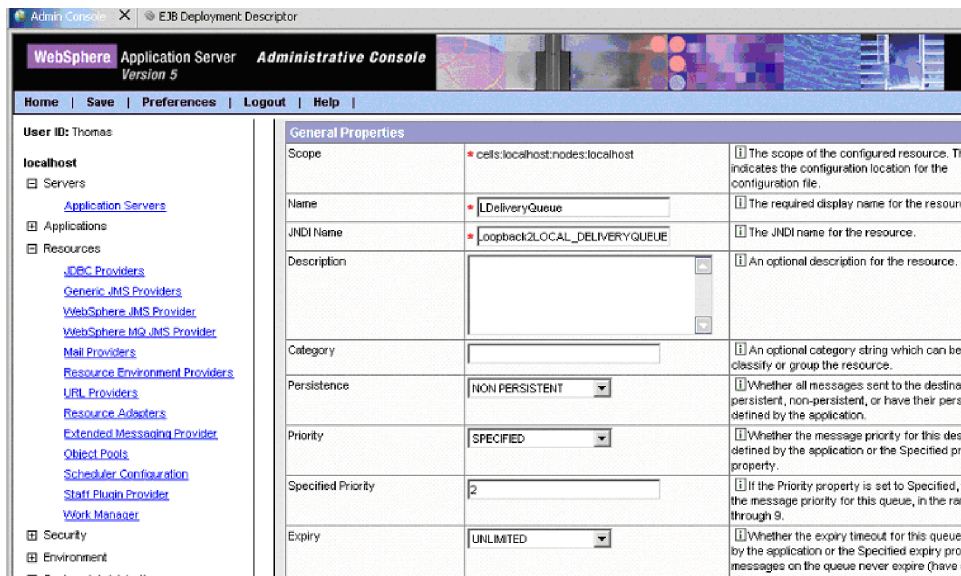


Figure 35. Providing queue display and JNDI lookup names

In the Base Queue Name field, enter the name of the physical queue that you created in the MQ queue manager (see Figure 36). This must correspond to the DeliveryQueue that you specified in the connector configuration properties using Connector Configurator.

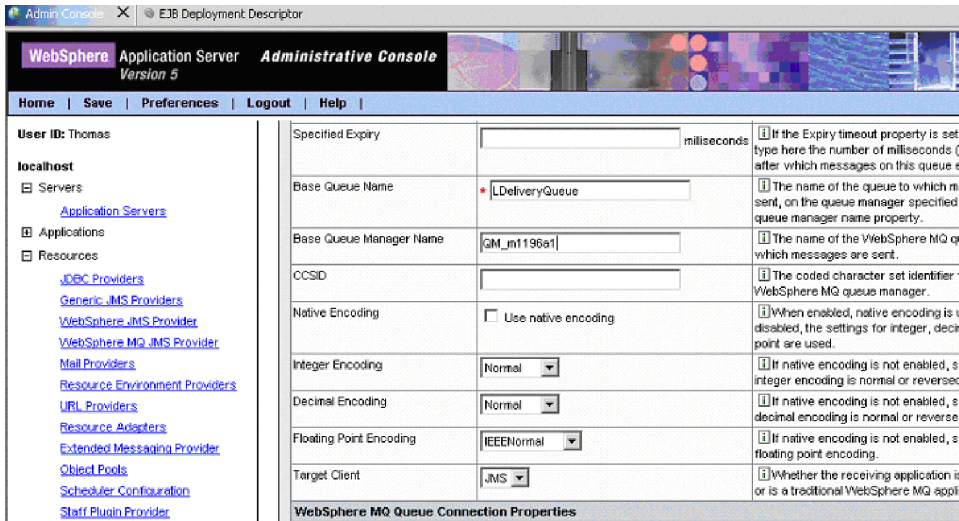


Figure 36. Entering the Base Queue Name

In the JNDI Name field, enter the name of the queue that you established in the destination JNDI queue name value in the Bean tab of EJB Deployment Descriptor. Provide a value for Base Queue Manager Name.

19. Create a queue connection factory, with the JNDI name corresponding to the name used in the EJB Deployment Descriptor and in the listener port.

After you have completed these steps, you will have a completed Enterprise Application Archive application for this interaction pattern. You can then test this Enterprise Application Archive application in the EE Test Environment and deploy the application as an Enterprise Application Archive file to another WebSphere Application Server environment.

Agent Request (Synchronous event delivery)

The Agent Request interaction pattern synchronously delivers a request message from the adapter to WebSphere Application Server and requires a response.

To create an Agent Request interaction pattern, you must create an Enterprise Application Archive application that contains:

- A message driven bean (MDB) that receives messages and invokes the EJB
- An EJB that is triggered by the MDB and that implements the business logic that you want to use for processing the event
- Helper classes that represent the .xsd files as java beans, and format handler java beans that convert XML to java, as specified in the .wsdl file.

The Enterprise Application Archive application must also contain .wsdl and .xsd files from the Service Project. If you chose Export to Service Project when you deployed your System Manager user project, the Service Project already exists and is included in the Enterprise Application Archive. If you instead exported your System Manager user project as individual files to a directory or as a JAR file, you will need to import the file or files into a new Service Project. To do so, follow the same procedure as described for an Agent Delivery interaction pattern, in “Import files for a new service project” on page 69.

The procedure for creating an Agent Request interaction pattern is similar to that for creating an Agent Delivery pattern, except that this pattern requires that a

response message be sent. Accordingly, the business logic in the EJB skeleton for the Agent Request interaction pattern must return a value.

Perform the following steps to create an Agent Request interaction pattern.

Create an EJB project

The Agent Request interaction pattern requires an EJB to consume the message incoming to WebSphere Application Server from the agent. You must both create the EJB and add business logic to the EJB method to process the received event.

Start by creating a project to contain the EJB:

1. In the J2EE view of the Business Integration perspective, choose File>New>Project.
2. In the New Project dialog, choose EJB in the left-hand panel, then select EJB project in the right-hand panel, and then choose Next.
3. In the Select an EJB Version dialog, choose Create 2.0 EJB Project, and click Next.
4. In the EJB Project Creation Dialog, provide a name for your EJB project (see Figure 37), and the name of a new or existing Enterprise Application Archive project folder to which it will be added. If you specify a name for a new Enterprise Application Archive project, a folder with that project name will be created.

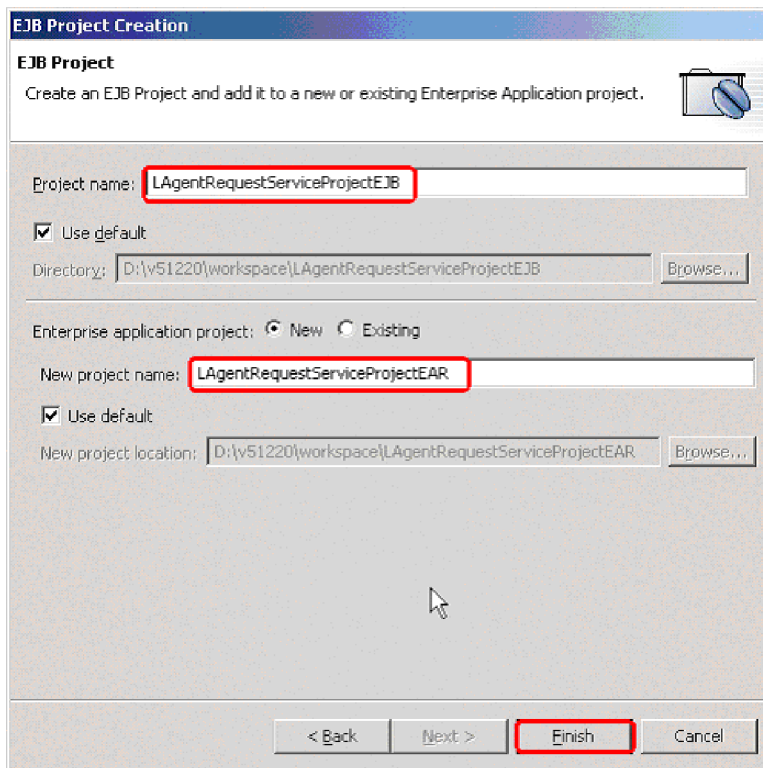


Figure 37. Naming the EJB project

5. Choose Finish. A project, containing an EJB module, is generated under the name that you specified. The EJB project is created for you and you can see it in the J2EE Hierarchy view under EJB Modules. In the Services view, under Service Projects, select the .wsdl interface file that you imported from your

- connector configuration (for example, AgentReqConnector.wsdl). Right click, and choose New>Build from Service. The New Service Skeleton dialog opens.
6. Choose EJB Service Skeleton (see Figure 27) and click Next.

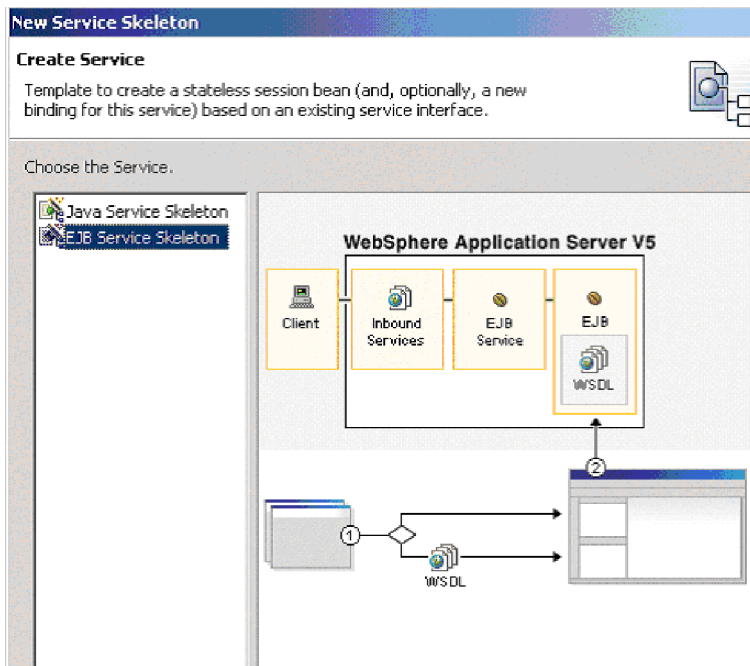


Figure 38. Choosing EJB Service Skeleton

7. In the New Service Skeleton dialog, select *Create a new port and binding* and select *Generate helper classes*. Choose Next.
8. The Service Skeleton dialog appears.
9. Select the appropriate Port type name according to the interaction pattern you are creating, and provide a meaningful package name in place of the default. Click Next.
10. In the next dialog, you can specify properties for generating the EJB skeleton, or accept the defaults. Click Finish. The newly created EJB project will be listed under Deployable Services in the Services view.
11. Edit the skeleton to add any necessary business logic.

Create an MDB

A message driver bean is responsible for receiving events as messages from the connector and invoking EJB processing of the events.

In this step, you will deploy the EJB service to the server using existing inbound port and bindings - the original JMS bindings. This step generates an MDB that invokes the appropriate method of the Session Bean skeleton. The MDB is specific to the port type and the selection of the operation is made based on the incoming message properties: WSDLOperation (and possibly WSDLInput and WSDLOutput) properties. The additional helper classes, format handlers, are also generated during the deployment. These are used to convert from the wire format (the XML message) to the Java Class, which in turn is passed as an argument to the business method of the EJB.

You must configure the JNDI names used by the EJB and MDB. The JNDI setup is necessary since the lookup names of the queue connection factory and queues defined in WSDL may not match the actual names defined on your server. The WebSphere Application Server allows you to map the name you are looking up (as specified in the .wsdl file) to the actual JNDI name deployed on the server, thus allowing correct execution without the need for modification of your wsdl files. You also need to specify what listener port is used by the MDB.

1. In the Package Explorer view, expand the folder for the Service Project that you have created and select the EJBService .wsdl file that was generated in the previous task. Right-click on the file, and in the pop-up choose Enterprise Services>Generate Deploy Code.
2. In the Generate Deploy Code dialog (see Figure 39), select *Use an existing port*. You can either select the same EJB project you created in the previous step (for the EJB skeleton) or specify the name of a new EJB project to create. A new EJB project must be in the same Enterprise Application Archive project as your existing EJB skeleton project. Choose Next.

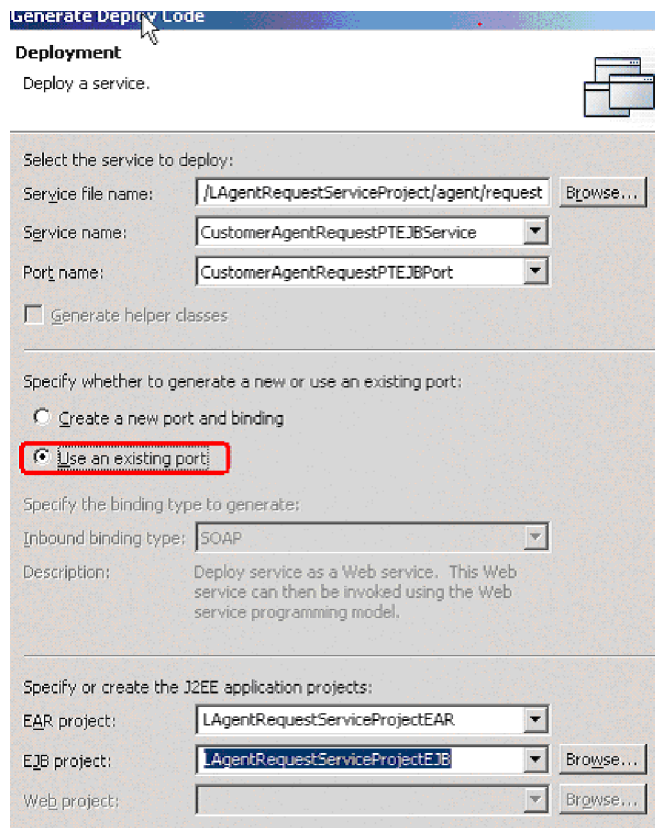


Figure 39. Generate Deploy Code dialog

3. In the Inbound Service Files dialog, select the JMSService .wsdl file for your project, and the corresponding service name and port name. Choose Finish.
4. In the J2EE Hierarchy view, select your project under EJB Modules, right-click, choose *Open With the EJB Deployment Descriptor* editor and choose the Bean tab. Verify that the Destination type is Queue. Specify the correct ListenerPort, so that its queue connection factory and queue match those specified in the .wsdl file for this interaction pattern.
5. Choose the References tab. Select the resource queue name listed under the MDB. The name from the .wsdl file then appears in the *Name* box at right.

Under WebSphere Bindings, in the *JNDI name* field, enter the correct JNDI name for the deployment of this queue, as defined in the WebSphere Application Server Administrative Console (see Figure 40).

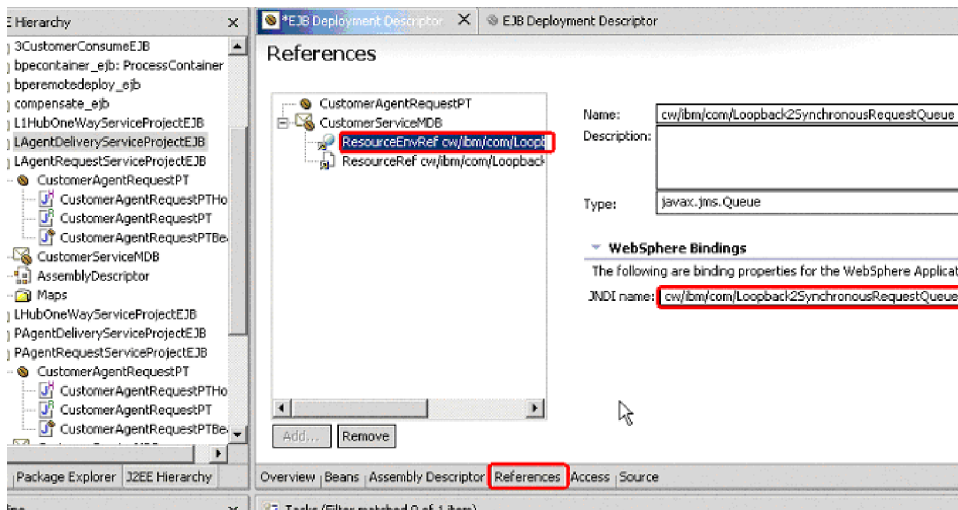


Figure 40. Entering the JNDI name for queue deployment

6. Select the connector factory listed under the MDB. Under WebSphere Bindings, in the *JNDI name* field, enter the correct JNDI name for the deployment of this connection factory, as defined in the WebSphere Application Server Administrative Console (see Figure 41).

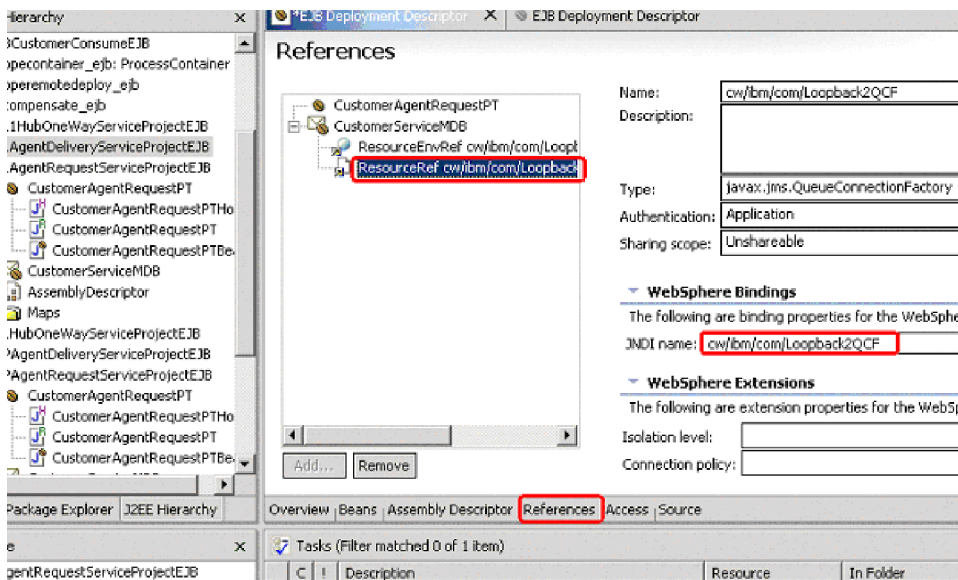


Figure 41. Entering the JNDI name for connection factory

7. Save the changes that you made in the EJB Deployment Descriptor editor.
8. Open the Server Configuration tab in the lower panel of the Business Integration perspective and right-click on the Servers icon.
9. Choose New, and choose Servers and Server Configuration. The *Create a New Server and Server Configuration* dialog appears.

10. In the Server name field, enter a name for the server instance you are creating. Enter the same name in the Folderfield. In the Server Type field, select EE Test Environment.
11. Choose Finish, and respond Yes when prompted to create a new server project with the name you specified.
12. After the utility completes, in the Server Configuration panel double-click the icon for the new server instance that you have created. The WebSphere Server panel appears, with the name of the server instance you have created. Choose the Configuration tab.
13. The WebSphere Server Configuration panel appears. Put a check in the Enable administration console check box.
14. Choose the Variables tab. In the Defined variables drop down box, locate the MQ_INSTALL_ROOT variable. Change the value of the variable to the absolute root path of your MQ installation.
15. Save your configuration. (Ctrl+S).
16. Start the newly configured server instances, as follows: In the lower portion of the Business Integration perspective, choose the Servers tab. The Servers panel displays. In the Servers panel, select the icon for the newly configured server instance, and click the run button to start the server.
17. Open the WebSphere Application Server Administrative Console. Under General Properties under the Configuration tab, add the listener port name that you specified in the Bean tab of EJB Deployment Descriptor (see Figure 42). Add the same connection factory JNDI value and destination JNDI queue name value that you specified in the References tab of EJB Deployment Descriptor. As appropriate for your needs, make any necessary changes in the Maximum sessions, Maximum retries, and Maximum messages fields.

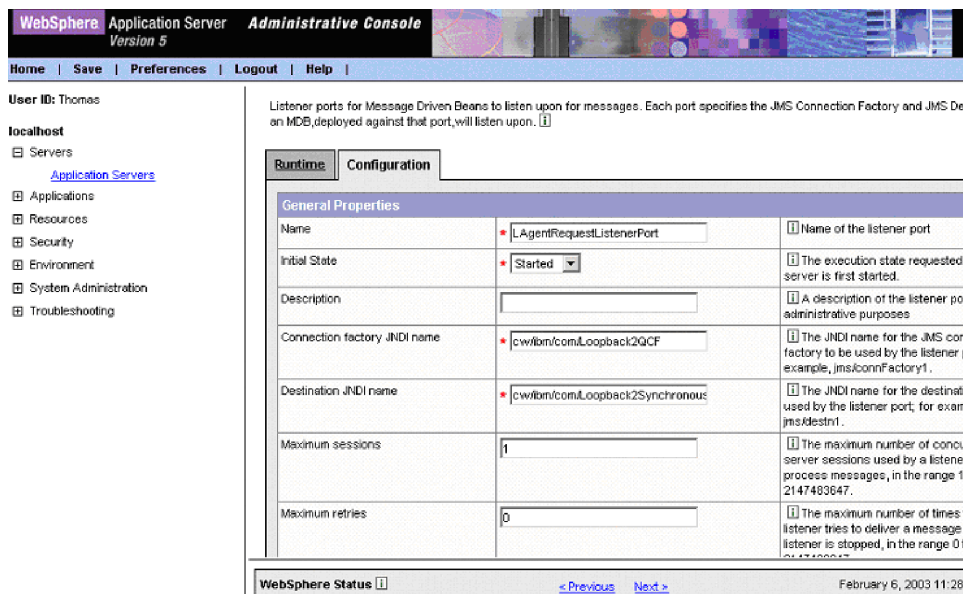


Figure 42. Adding the Listener Port and JNDI names

18. In the WebSphere Application Server Administrative Console, add the queue as a resource (see Figure 43). Provide a display name for the queue, and use that same value in the Base Queue Name field. In the JNDI Name field, enter the name of the queue that you established in the destination JNDI queue name value in the Bean tab of EJB Deployment Descriptor.

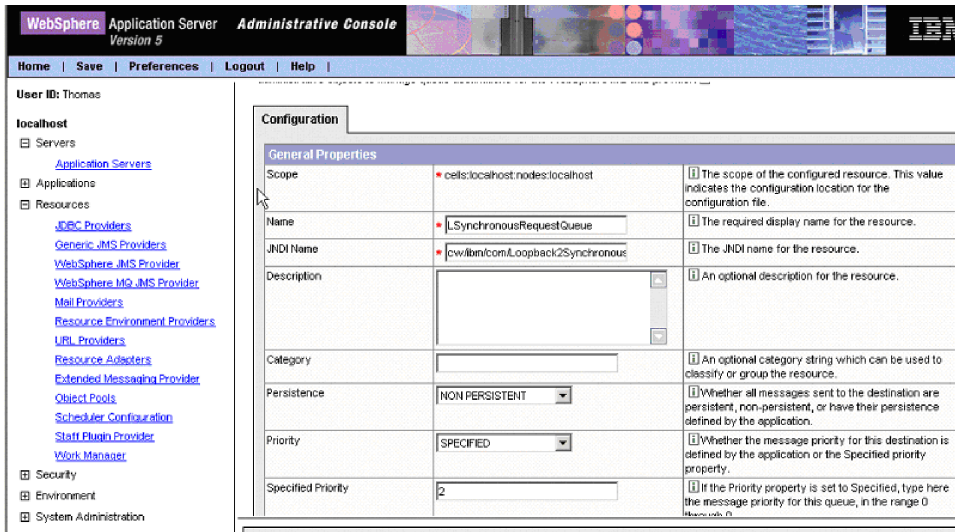


Figure 43. Adding the queue as a resource

Provide a value for Base Queue Manager Name (see Figure 44).

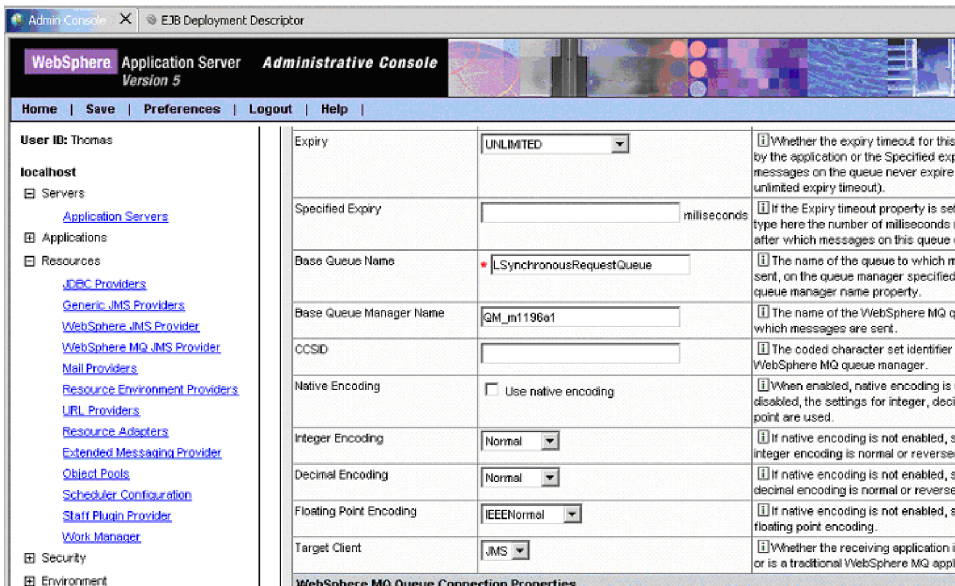


Figure 44. Providing a Base Queue Manager name

After you have completed these steps, you will have a completed Enterprise Application Archive application for this interaction pattern. You can then test this Enterprise Application Archive application in EE Test Environment and deploy the Enterprise Application Archive application to another WebSphere Application Server environment.

Hub One Way

The Hub One Way interaction pattern delivers a message from WebSphere Application Server to the adapter, without expecting a response from the adapter.

The Enterprise Application Archive application for the interaction pattern must contain a service project. If you chose Export to Service Project when you deployed

your System Manager user project, the service project already exists. If you exported the System Manager user project as a JAR file or as a set of files to a directory, you will need to import the files to create a service project. If you need to perform that task, see “Import files for a new service project” on page 69, which describes that procedure for the Agent Delivery interaction pattern. The remainder of these instructions for the HubOneWay interaction pattern assume that you chose Export to Service Project when you deployed your System Manager user project.

To create a HubOneWay interaction pattern, create an Enterprise Application Archive application that contains:

- The Session Bean that invokes the JMS service and can be used from any J2EE application. The Session Bean has a business method for each operation of the deployed (JMS) Port Type.

After creating the Session Bean, you must configure it, using the Deployment Descriptor editor, to perform necessary WebSphere mappings between JNDI names and the queue names and connection factory names derived from the .wsdl file.

- Helper classes that represent the .xsd files as java beans, and format handler java beans that convert XML to java, as specified in the .wsdl file.

These tasks are described in the topics that follow.

Creating the EJB

This step creates a Session Bean, along with the .wsdl files describing it (the EJB service and bindings).

1. In the Package Explorer view, select the JMSServices .wsdl file. (Note that because the service project is identical for all four interaction patterns, this can be the same JMSServices file generated when you created the service project for the Agent Delivery interaction pattern.)
2. Right-click the file and choose Enterprise Services>Generate Deploy Code. The Generate Deploy Code dialog displays (see Figure 45).

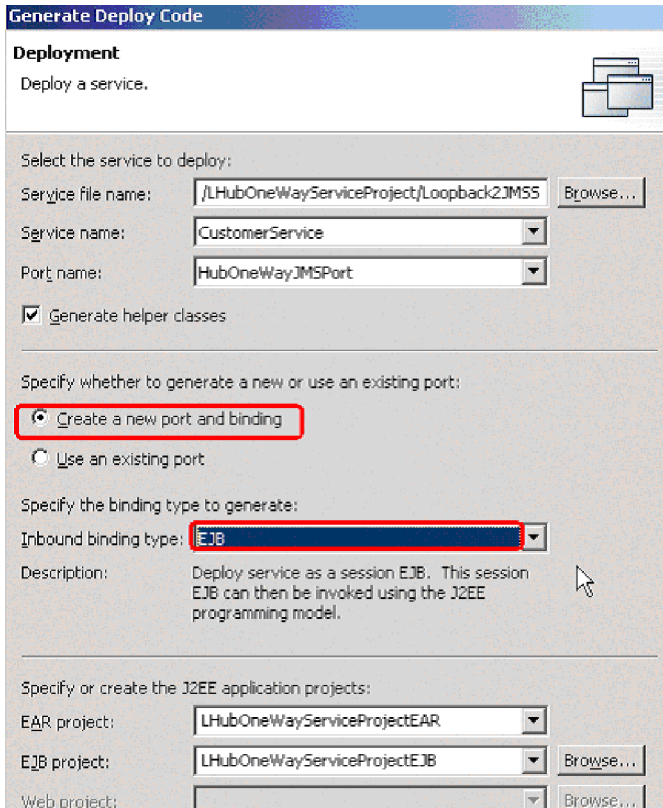


Figure 45. Generate Deploy Code for hub one way

3. In the Port Name drop-down, select the port name for this interaction pattern.
4. Select *Create a new port and binding*.
5. In the *Inbound binding type* drop-down, select EJB, JMS, or SOAP. Your selection affects the WSDL files generated, for the EJB access you get the EJB Bindings and Service files.
6. Choose Next. The Inbound Service Files dialog appears.

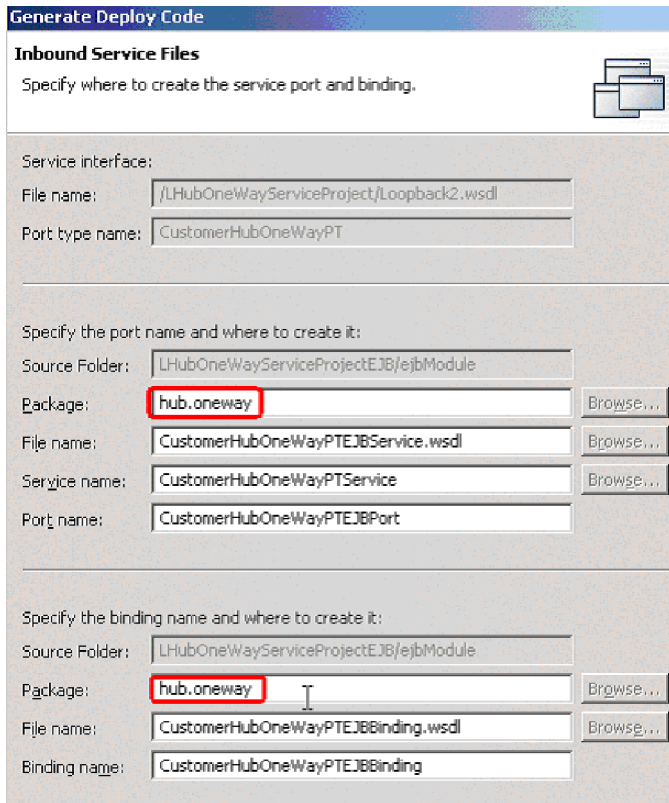


Figure 46. Inbound Service Files dialog

7. In the Inbound Service Files dialog, specify a meaningful package name for both the port name and the binding name.
8. Choose Next. The EJB Port dialog appears, with a field that specifies a JNDI name as an EJB port property. Accept the default value.
9. Use the Deployment Descriptor editor to map JNDI names to a queue connection factory name and a queue name (see Figure 47 and Figure 48).

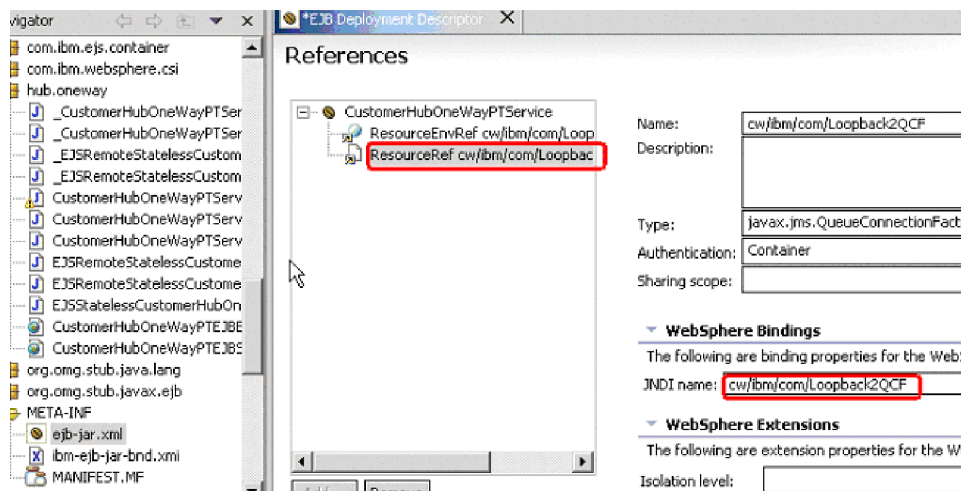


Figure 47. Mapping JNDI names, part 1

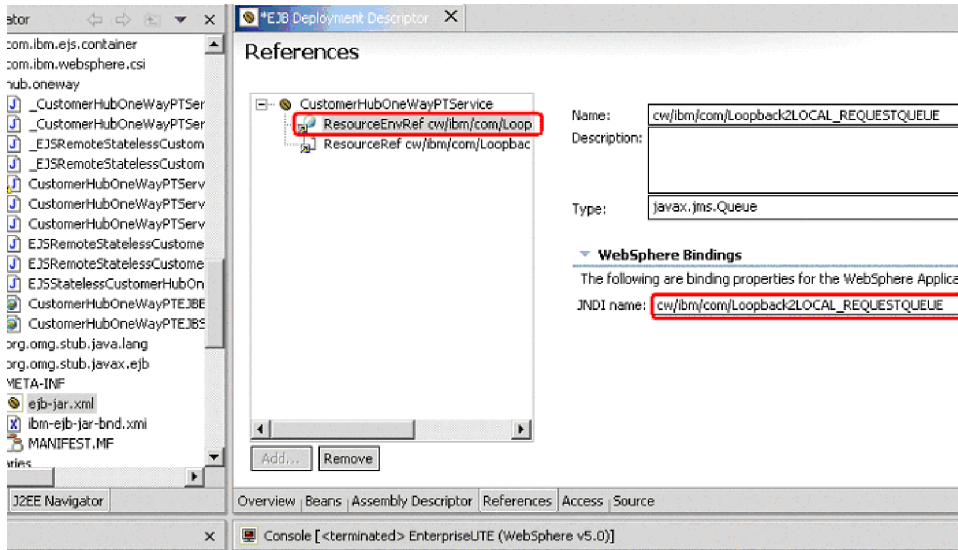


Figure 48. Mapping JNDI names, part 2

10. An EJB that represents Hub One Way is completed. You can deploy it or use it as any other WebSphere Application Server component.

Create an application client project for testing

To be able to test the complete application package, you may optionally choose to add a client project to the Enterprise Application Archive file. Create the Enterprise Application Archive Client project with the client class. In the main() method of the class, the code should do the lookup of the EJB and invoke its appropriate business method. Before the application client can be run, you will need to specify the name of the class containing the main() method of the client in the Application Client deployment descriptor.

To create the application client project:

1. In the Package Explorer view, choose File>New project and choose J2EE>Application Client Project. Choose Next.
2. In the J2EE Specification version dialog, choose Create J2EE 1.3 Application Client project. Choose Next.
3. In the Application Client project creation dialog, provide a project name, check Existing, and browse to select the name of the source Enterprise Application Archive project containing the EJB you created in the previous task. Choose Next.
4. The Module Dependencies dialog appears (see Figure 49), with a list of available dependent JAR files. Mark the boxes for the JAR files that should be included in the Enterprise Application Archive project with this module, and choose Finish.

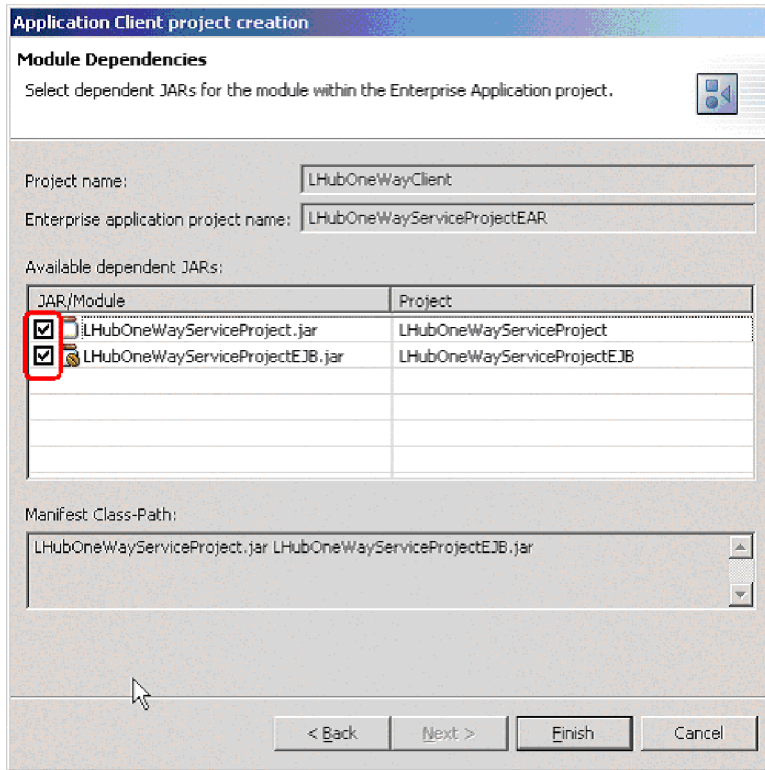


Figure 49. Module Dependencies dialog

5. To create a package within the client application, in the Package Explorer view, select the application client module that you have created, and choose the Create a Java Package button from the toolbar. The New Java Package dialog appears. Enter a name for the package and choose Finish. A folder with that package name is added under the application client module folder.
6. To create a Java class in which to add logic, select the application client folder and choose the Create Java Class button from the menu bar.
7. The New Java Class dialog appears (see Figure 50). In the Package field, select the client package that you created previously; provide a name for the class; specify Public for modifiers; and specify public static void and inherited abstract methods for the method stubs to be created.

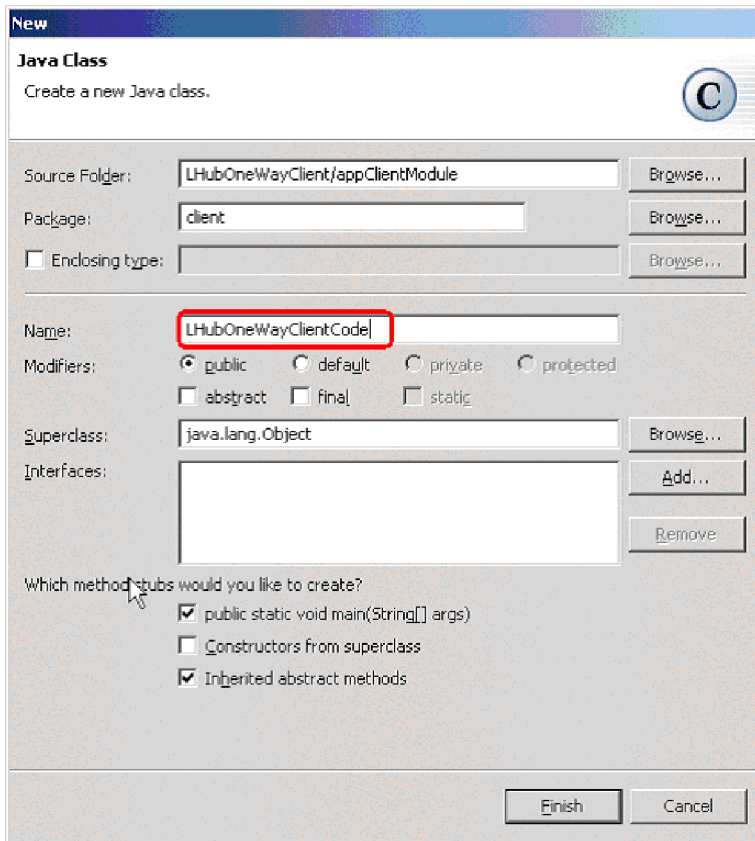


Figure 50. New Java Class dialog

8. Choose Finish.
9. Open the Java class and add your logic.
10. Add necessary Jar files. To do so, in Package Explorer select and right-click the application client, choose Properties, and choose Java Build Path. The Java Build Path panel displays.
11. Choose the Libraries tab, choose Add External Jars, and navigate to the directory in which you installed WebSphere Studio Application Developer Integration Edition, and expand the \runtime\ee_v5\lib folder. Select the files, including wsatlib.jar, wsdl4j.jar, wsif.jar, qname.jar (see Figure 51).

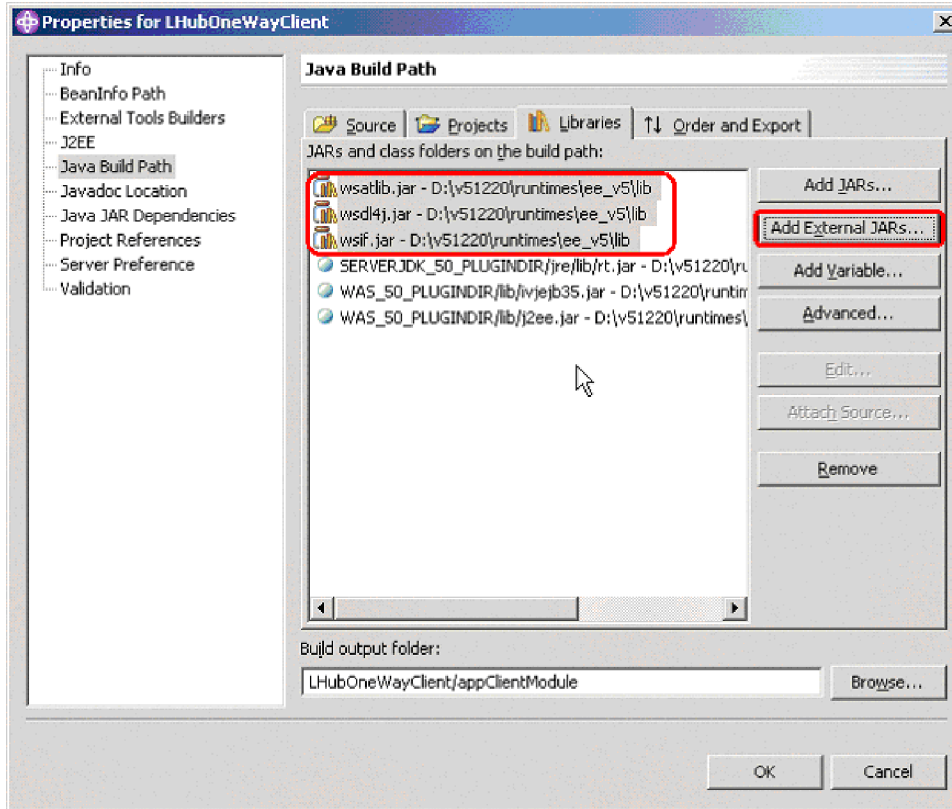


Figure 51. Selecting jar files

12. Click OK.
13. Update the application client package using the Deployment Descriptor Editor. To do so: In Package Explorer, expand the META-INF folder under the application client module, and select the application-client.xml file. Right click the file, and choose Open With>Deployment Descriptor Editor. The Client Deployment Descriptor dialog appears (see Figure 52).

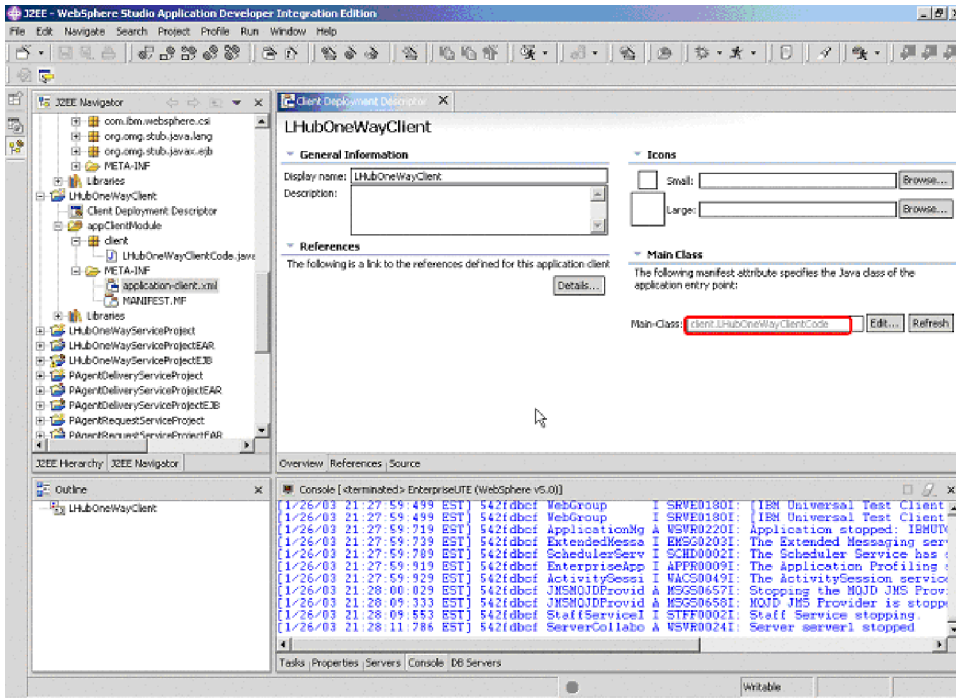


Figure 52. Client Deployment Descriptor dialog

14. Add the Main-Class name. The Application client is ready to use.

Hub Request

The Hub Request interaction pattern synchronously delivers a message from WebSphere Application Server to the adapter and receives a reply.

The Enterprise Application Archive application for the interaction pattern must contain a service project. If you chose Export to Service Project when you deployed your System Manager user project, the service project already exists. If you exported the System Manager user project as a JAR file or as a set of files to a directory, you will need to import the files to create a service project. If you need to perform that task, see “Import files for a new service project” on page 69, which describes that procedure for the Agent Delivery interaction pattern. The remainder of these instructions for the Hub Request interaction pattern assume that you chose Export to Service Project when you deployed your System Manager user project.

To create a Hub Request interaction pattern, create an Enterprise Application Archive application that contains:

- The Session Bean that invokes the JMS service and can be used from any J2EE application. The Session Bean has a business method for each operation of the deployed (JMS) Port Type.
After creating the Session Bean, you must configure it, using the Deployment Descriptor editor, to perform necessary WebSphere mappings between JNDI names and the queue names and connection factory names derived from the .wsdl file.
- Helper classes that represent the .xsd files as java beans, and format handler java beans that convert XML to java, as specified in the .wsdl file.

These tasks are described in the topics that follow.

Creating the EJB project

This step creates a Session Bean, along with the .wsdl files describing it as an EJB service and bindings.

1. In the Package Explorer view, select the JMSServices .wsdl file. (Note that because the service project is identical for all four interaction patterns, this can be the same JMSServices file generated when you created the service project for the Agent Delivery interaction pattern.)
2. Right-click the file and choose Enterprise Services>Generate Deploy Code (see Figure 53). The Generate Deploy Code dialog displays.

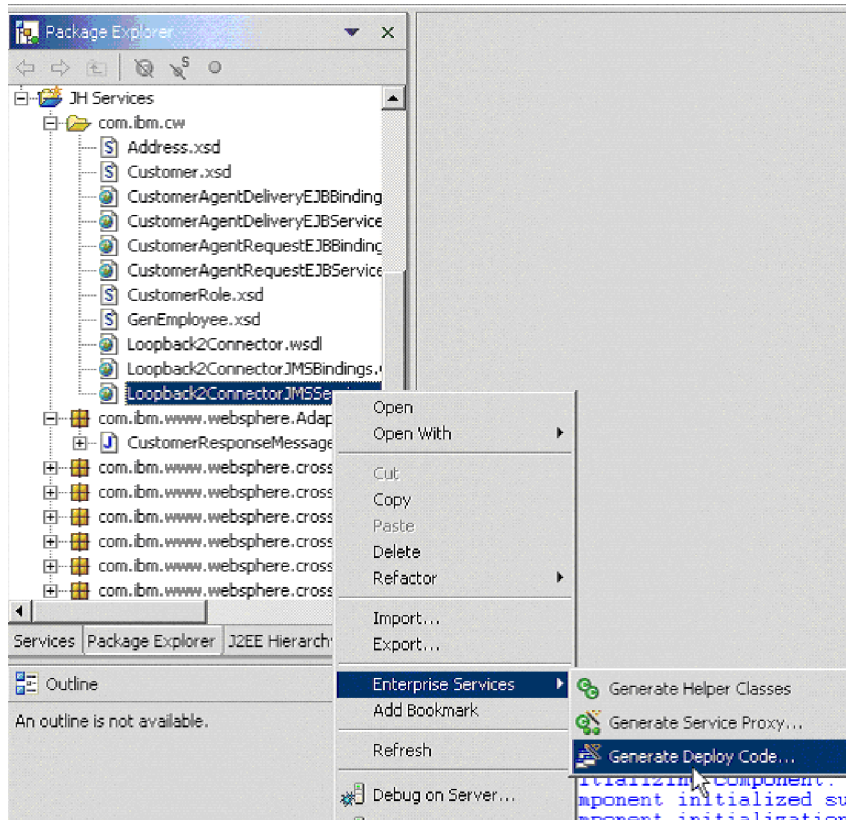


Figure 53. Choosing Generate Deploy Code

3. In the Port Name drop-down, select the port name for this interaction pattern.
4. Select *Create a new port and binding* (see Figure 54).

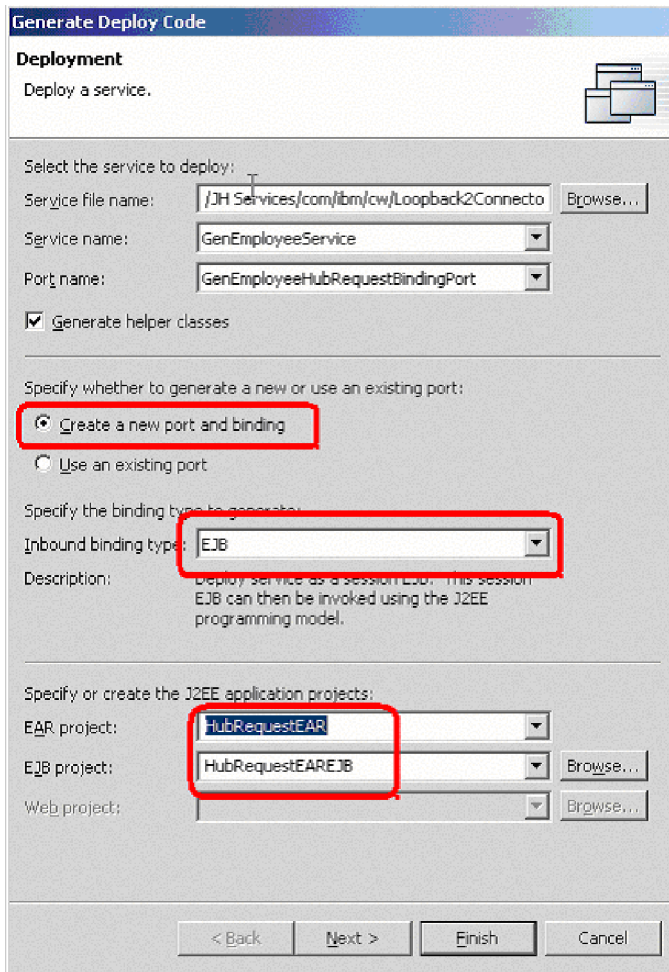


Figure 54. Selecting Create a new port and binding

5. In the *Inbound binding type* drop-down, select EJB, JMS, or SOAP. Your selection affects the WSDL files generated, for the EJB access you get the EJB Bindings and Service files.
6. Choose Next. The Inbound Service Files dialog appears.
7. In the Inbound Service Files dialog, edit as necessary and specify a meaningful package name for both the port name and the binding name.
8. Choose Next if you need to change the JNDI lookup name, or choose Finish.
9. Use the Deployment Descriptor editor to map queue names and queue connection factory names, in the same manner as you did for the Hub One Way interaction pattern. But for the Hub Request interaction pattern to execute correctly, the sending and receiving of the message must not be part of the same transaction. The simple solution is to disable transactions on the EJB in its deployment descriptor (see Figure 55):

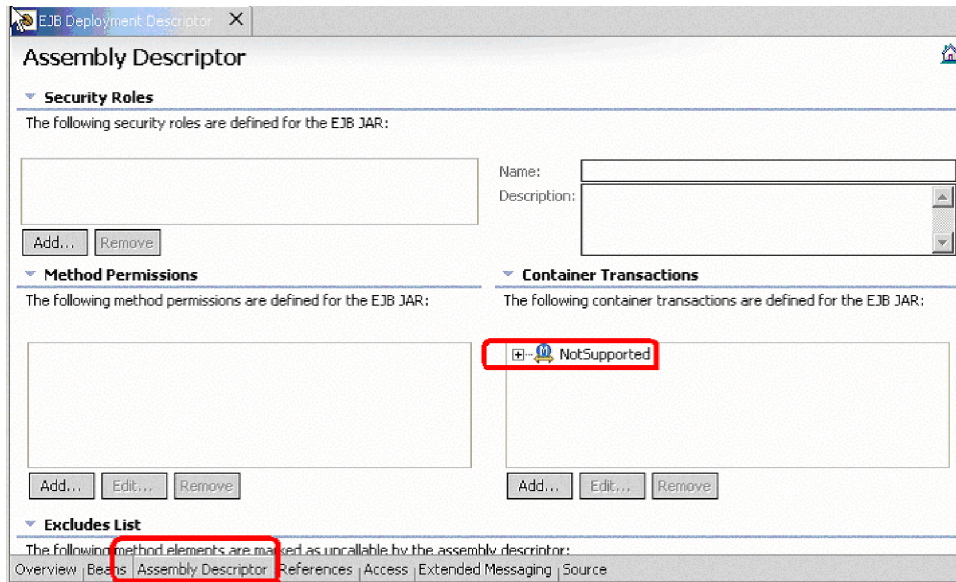


Figure 55. Mapping the Hub Request interaction pattern

10. An EJB that represents Hub Request is completed. You can deploy it or use it as any other WebSphere Application Server component.

Optionally, after completing the above steps, you can create an application client project for testing. See the steps describing this task for the Hub One Way interaction pattern.

Requirements for initiating business objects in interaction patterns

The topics in this section describe mandatory requirements for the use of business objects in interaction patterns.

Adding the business object initialization library

The BusinessObject initialization library routine is required for EJBs that create business objects. The JAR file containing it (boutil.jar) must be part of the Enterprise Application Archive project and specified as the dependency of the EJB module. These are the steps:

1. Select the Enterprise Application Archive application in the Business Integration Perspective, J2EE Hierarchy View. Right-click on it and choose Import.
2. Use the Import dialog to navigate to and import the boutil.jar file that is shipped with the adapter.
3. Verify that the newly imported file appears within the folder hierarchy for the enterprise application:
4. Add the imported jar to the list of dependencies of the EJB module (in the above Enterprise Application Archive) in which you are planning to use the utility library. Select the EJB project, then properties from the popup menu and Java JAR Dependencies in the list (see Figure 56).

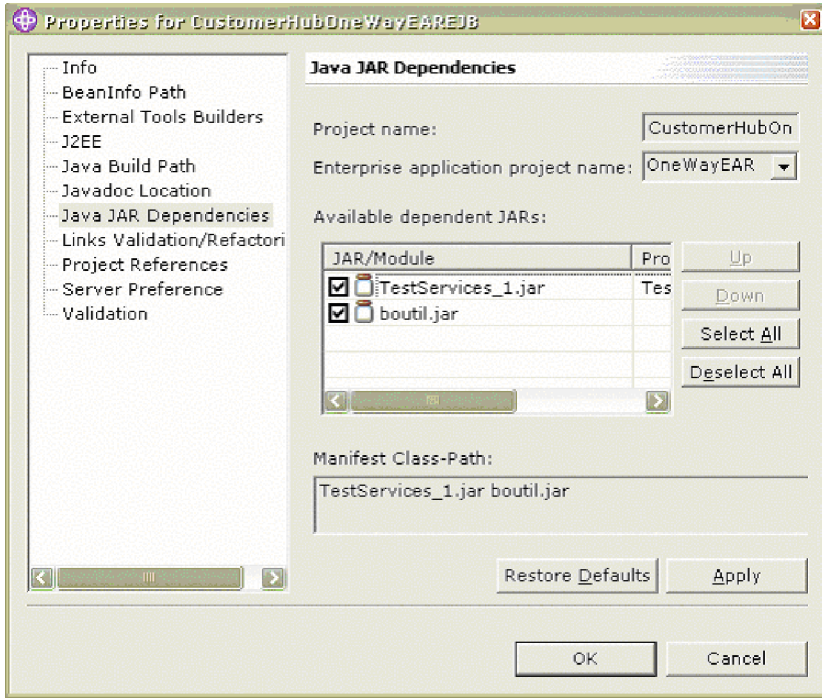


Figure 56. Selecting java jar dependencies

5. Check the box for boutil.jar
6. Press OK.

Now the boutils.jar file is part of the Enterprise Application and will be included in the Enterprise Application Archive file.

Reserved values in business object handling

At runtime, when business objects containing the CxIgnore or CxBlank attribute values are sent from a source adapter to the WebSphere Application Server broker implementation, the CwXML data handler sets reserved values in place of those attribute values in the XML instance document that it passes to the broker; after the broker processes the business object and returns it as an XML message to the destination adapter, the data handler in the destination adapter restores the CxIgnore and CxBlank values. This sequence of actions is necessary to prevent the java object in the broker from assigning a default value of zero to those attributes when it processes the business object.

The following table shows the reserved values that are used in place of the values CxIgnore and CxBlank for each data type:

Data type	CxIgnore	CxBlank
int	Integer.MIN_VALUE	Integer.MAX_VALUE
float	Float.MIN_VALUE	Float.MAX_VALUE
double	Double.MIN_VALUE	Double.MAX_VALUE
string, date, longtext	"CxIgnore"	

Note: For the Boolean data type, possible values are either 'true' or 'false'. Due to this limitation, it is not possible to set CxIgnore or CxBlank values for

Boolean data type. The procedure for handling the Boolean data type is to define the attribute type as String data type to set 'CxIgnore' or 'CxBlank' values.

Business objects sent from the adapter agent are initiated to the values shown in the table above. However, any business object created in the user application, as in a Hub One Way or Hub Request interaction pattern, must be explicitly initiated. This can be accomplished by invoking `BusinessObjectUtilities.initializeBO(<BOInstance>)` on every business object created, as in the following sample fragment:

```
TestCustomerElement customer = new TestCustomerElement();
    AddressElement address = new AddressElement();
        customer.setAddress(address);
    BusinessObjectUtilities.initializeBO(customer);
    customer.setCustomerId("2424234");
    address.setCountry("USA");
```

Note that you should create all your business objects, create their parent/child structure, and invoke the initialization routine on the parent business object, before any business object attributes are set.

Transactional support

Transactional behavior for all interaction patterns is supported with the following conditions.

In the HubRequest pattern, the sending of the message and waiting for and receiving of the response must not be part of one transaction. If one transaction is used, the J2EE component will block indefinitely, since no message can be received until one is sent, which only happens when the transaction is committed.

A special care must be taken when transactions are used with the patterns that originate from the agent (AgentDelivery and AgentRequest) to prevent disabling of the message listener. When the MDB receives an agent-originated message and an exception is thrown in MDB or any other WebSphere Application Server component it invokes, the transaction is rolled back and the message is put back in the queue and then redelivered. If the error continues, the redelivery either stops after a predefined number of attempts, disabling the listener, or continues indefinitely (depending on the listener port setting). To prevent this situation, if transactions are used, the user should add user code to the MDB to detect and handle the redelivered messages. The sample below shows how the redelivered message can be detected in the MDB.

```
public void onMessage(javax.jms.Message msg) {
    try {
// -> new code start
        if(msg.getJMSRedelivered()){
            System.out.println("Message redelivered");
            // User code to handle redelivered message
        }
        else{
            System.out.println("Message delivered first time");
            // First time delivery, unless user code for
            // redelivery returns, it should invoke
            executeOperation(msg)
        }
// <- new code end
        executeOperation(msg);
    }
    catch (WSIFException e) {
        e.printStackTrace();
    }
}
```

```
    }  
    catch (javax.jms.JMSEException e) {  
        e.printStackTrace();  
    }  
}
```

Chapter 9. Administering the business integration system

This chapter provides information about the administrative tasks you need to perform for WebSphere Business Integration adapters.

The following sections are included in this chapter:

- “Starting a connector”
- “Stopping a connector” on page 106
- “Managing log and trace files” on page 108
- “Using Log Viewer to view connector messages” on page 115

Starting a connector

A connector must be explicitly started using its **connector start-up script**. On Windows systems the startup script should reside in the connector’s runtime directory:

```
ProductDir\connectors\connName
```

where *connName* identifies the connector.

On UNIX systems the startup script should reside in the *ProductDir/bin* directory.

The name of the startup script depends on the operating-system platform, as Table 10 shows.

Table 10. Startup scripts for a connector

Operating system	Startup script
UNIX-based systems	connector_manager
Windows	start_connName.bat

When the startup script runs, it expects by default to find the configuration file in the *Productdir* (see the commands below). This is where you place your configuration file.

Note: You need a local configuration file if the adapter is using JMS transport.

You can invoke the connector startup script in any of the following ways:

- On Windows systems, from the **Start** menu
Select **Programs>IBM WebSphere Business Integration Adapters>Adapters>Connectors**. By default, the program name is “IBM WebSphere Business Integration Adapters”. However, it can be customized. Alternatively, you can create a desktop shortcut to your connector.

- From the command line

- On Windows systems:

```
start_connName connName brokerName [-cconfigFile ]
```

- On UNIX-based systems:

```
connector_manager -start connName brokerName [-cconfigFile ]
```

where *connName* is the name of the connector and *brokerName* identifies your integration broker, as follows:

- For WebSphere InterChange Server, specify for *brokerName* the name of the ICS instance.
- For WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, or WebSphere Business Integration Message Broker) or WebSphere Application Server, specify for *brokerName* a string that identifies the broker.

Note: For a WebSphere message broker or WebSphere Application Server on a Windows system, you *must* include the `-c` option followed by the name of the connector configuration file. For ICS, the `-c` is optional.

- From Adapter Monitor (available only when the broker is WebSphere Application Server or InterChange Server), which is launched when you start System Manager
You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Manager (available for all brokers)
You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector starts when the Windows system boots (for an Auto service) or when you start the service through the Windows Services window (for a Manual service).

For more information on how to start a connector, including the command-line startup options, refer to one of the following documents:

- For WebSphere InterChange Server, refer to the *System Administration Guide*.
- For WebSphere message brokers, refer to *Implementing Adapters with WebSphere Message Brokers*.
- For WebSphere Application Server, refer to *Implementing Adapters with WebSphere Application Server*.

Stopping a connector

The way to stop a connector depends on the way that the connector was started, as follows:

- If you started the connector from the command line, with its connector startup script:
 - On Windows systems, invoking the startup script creates a separate “console” window for the connector. In this window, type “Q” and press Enter to stop the connector.
 - On UNIX-based systems, connectors run in the background so they have no separate window. Instead, run the following command to stop the connector:
`connector_manager_connName -stop`
where *connName* is the name of the connector.
- From Adapter Monitor (WebSphere Business Integration Adapters product only), which is launched when you start System Manager
You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Monitor (WebSphere InterChange Server product only)
You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.

- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector stops when the Windows system shuts down.

Creating multiple connector instances

Creating multiple instances of a connector is in many ways the same as creating a custom connector. You can set your system up to create and run multiple instances of a connector by following the steps below. You must:

- Create a new directory for the connector instance
- Make sure you have the requisite business object definitions
- Create a new connector definition file
- Create a new start-up script

Create a new directory

You must create a connector directory for each connector instance. This connector directory should be named:

`ProductDir\connectors\connectorInstance`

where `connectorInstance` uniquely identifies the connector instance.

If the connector has any connector-specific meta-objects, you must create a meta-object for the connector instance. If you save the meta-object as a file, create this directory and store the file here:

`ProductDir\Repository\connectorInstance`

Create business object definitions

If the business object definitions for each connector instance do not already exist within the project, you must create them.

1. If you need to modify business object definitions that are associated with the initial connector, copy the appropriate files and use Business Object Designer to import them. You can copy any of the files for the initial connector. Just rename them if you make changes to them.
2. Files for the initial connector should reside in the following directory:

`ProductDir\Repository\initialConnectorInstance`

Any additional files you create should be in the appropriate `connectorInstance` subdirectory of `ProductDir\Repository`.

Create a connector definition

You create a configuration file (connector definition) for the connector instance in Connector Configurator. To do so:

1. Copy the initial connector's configuration file (connector definition) and rename it.
2. Make sure each connector instance correctly lists its supported business objects (and any associated meta-objects).
3. Customize any connector properties as appropriate.

Create a start-up script

To create a startup script:

1. Copy the initial connector's startup script and name it to include the name of the connector directory:
`dirname`

2. Put this startup script in the connector directory you created in “Create a new directory” on page 107.
3. Create a startup script shortcut (Windows only).
4. Copy the initial connector’s shortcut text and change the name of the initial connector (in the command line) to match the name of the new connector instance.

You can now run both instances of the connector on your integration server at the same time.

For more information on creating custom connectors, refer to the *Connector Development Guide for C++ or for Java*.

Clearing messages from WebSphere MQ queues

WebSphere Business Integration adapters provide a sample batch file you can use to clear messages from the WebSphere MQ queues in the business integration system. Clearing the queues might be necessary if a problem with the business integration system prevents messages from being removed for processing.

To clear messages from WebSphere MQ queues, run the batch file, `clear_mq.bat` (Windows) or `clear_mq` (UNIX) located in the `ProductDir\templates` directory. This batch file clears messages from the queues specified in the file `crossworlds_mq.tst`. For more information about editing `crossworlds_mq.tst`, see “Using WebSphere Business Integration adapter batch files to configure WebSphere MQ queues” on page 54.

Managing log and trace files

The following tools provide graphical user interfaces for configuring and viewing message logging and tracing. Use the:

- Connector Configurator to set up or change connector logging, and tracing
- Log Viewer to display log and trace files.

In addition to using Log Viewer to view logs, you can open the log with a text editor.

During startup, the connector generates a temporary log file. This file contains all messages that are logged during startup, including connector properties and business object definitions that are passed to the connector framework. The file name is `broker_name_connector_name_tmp.log`, and it is written to the `ProductDir` directory. Once the connector is running, it handles logging and tracing as configured in the standard connector configuration properties.

While logging and tracing messages are written using UTF-8 encoding in the locale specified in the connector’s configuration file, time information is written in a locale-independent format.

At connector startup, a log file is also created, if one does not already exist, in the location specified in the connector’s configuration file. If a log file already exists, new entries are simply added to it. Unless a limit has been placed on the size of the connector log file, its size depends on the amount of time since it was last managed and the volume of transactions passing through the system. If the connector log file is configured with no size limit, it can continue to grow until it cannot be opened or it exhausts its disk space.

If tracing is enabled, a trace file is created at startup if one does not already exist. The size of the trace file must be managed in the same manner as the connector log file to avoid the problems caused by excessively large files.

Table 8 on page 61 lists the files used by the connector to store logging and tracing information.

To manage log and trace files, you can use the Connector Configurator to:

- Specify a size limit for log and trace files
- Have the files automatically archived once they reach their size limit.
- Specify the number of archive files to maintain.

For more information about using the Connector Configurator to set these options, see “Configuring logging and tracing options” on page 61.

Archival logging of log and trace files

If archival logging is enabled, each time the connector’s log or trace file reaches its maximum size, it is renamed as a new archive file. The archive file’s name is derived from the original log or trace file name, with the following inserted into the name: *_Arc_number*.

For example, if five archive files are to be used and the log file has the name `Connector.log`, then:

- The first archive created is named `Connector_Arc_01.log`.
- When the new log file fills up, `Connector_Arc_01.log` is renamed `Connector_Arc_02.log`,
- New log information is again saved to `Connector_Arc_01.log` and so on in a circular fashion, until there are five archive files.
- If there are already five archive files when a new log file is created, the oldest one, number five, is deleted. Then the remaining archive files are renamed and their numbers incremented so the number of archives matches the number you configured. Figure 57 on page 110 shows the progression of files using this configuration.

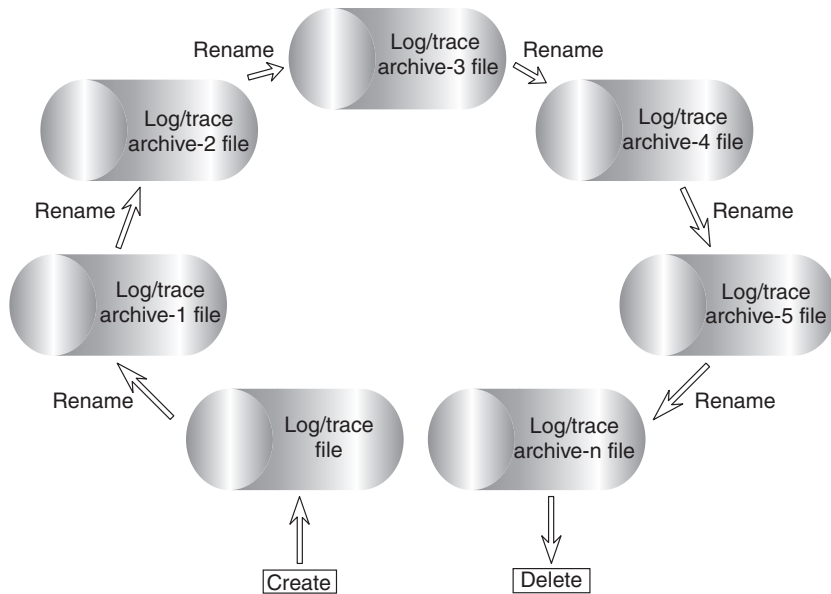


Figure 57. Circular archival logging.

See the configuration task “Configuring logging and tracing options” on page 61 for details.

Managing other files

While log and trace files can be managed by means of archival logging, other log files specific to each application need to be managed manually. Most of these files are created during runtime if they do not already exist. New information is appended to any existing file.

Any file management procedure can be used, but IBM suggests the following periodic log file management:

- Rename the files by appending a date to the file.
- Move the files to an archive directory.

Using Adapter Monitor and Fault Queue Manager

Adapter Monitor and Fault Queue Manager allow you to change the state of an adapter and to handle failed events that have been received by the fault queue. For more information on using System Manager, refer to the *IBM WebSphere Interchange Server System Administration Guide*.

Note: In this section the term “adapter” means the same thing as the term “connector” in other WebSphere Business Integration adapters documentation. References in this section to “changing the state of an adapter” mean the same thing as “changing the state of a connector.”

Adapter Monitor perspective

The Adapter Monitor perspective enables you to administer adapters and, through the Fault Queue Manager panel, to resubmit messages when errors occur in the processing of submitted events.

Adapter Monitor is used only with adapters that have been configured for use with JMS.

Note: Adapter Monitor runs in a Windows environment. To use Adapter Monitor for adapters installed on UNIX, you must create remote queues.

Opening Adapter Monitor

You can open Adapter Monitor in either of the following ways:

- From the drop-down menu in WebSphere Studio Application Developer Integration Edition, choose **Window>Open Perspective>Adapter Monitor**.
- In the System Manager perspective, expand a WebSphere Application Server project folder, right-click on a connector definition icon, and choose **Adapter Monitor** from the pop-up dialog.

In either method, the Adapter Monitor window displays:

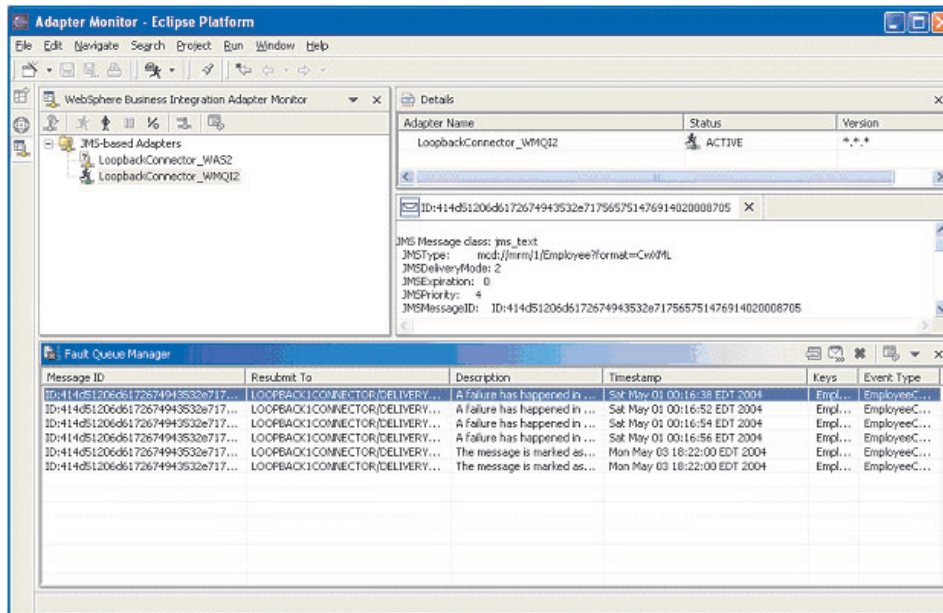


Figure 58. Adapter Monitor perspective

Setting Adapter Monitor preferences

You can set preferences that determine the intervals at which the Adapter Monitor will poll the state of the adapter and the number of messages that will be displayed in the Fault Queue Manager. To set the preferences, from the drop-down menu in WebSphere Studio Application Developer Integration Edition, choose **Window>Preferences>Adapter Monitor Preferences**, and set values for the following:

- **Adapter Monitor View** Enter a numeric value for the number of seconds that will elapse between each poll of the adapter status. The default is 30 seconds, the minimum elapsed time is 10 seconds, the maximum is 5 minutes.
- **Fault Queue Manager** Enter a numeric value for the maximum number of messages that will be displayed by Fault Queue Manager. The default is 10 messages, the minimum is 1, and the maximum is 50.

Choose **Apply** or **OK**.

Loading an adapter

There are two ways to load an adapter:

- From System Manager: Locate the adapter in the User Projects of the System Manager perspective. Right click and choose **Adapter Monitor** from the context menu. The Adapter Monitor perspective is opened, if not already open, and the adapter is loaded into Adapter Monitor.
- From Adapter Monitor: Right click **JMS-based Adapters** and choose **Load Adapter...** The following dialog appears:

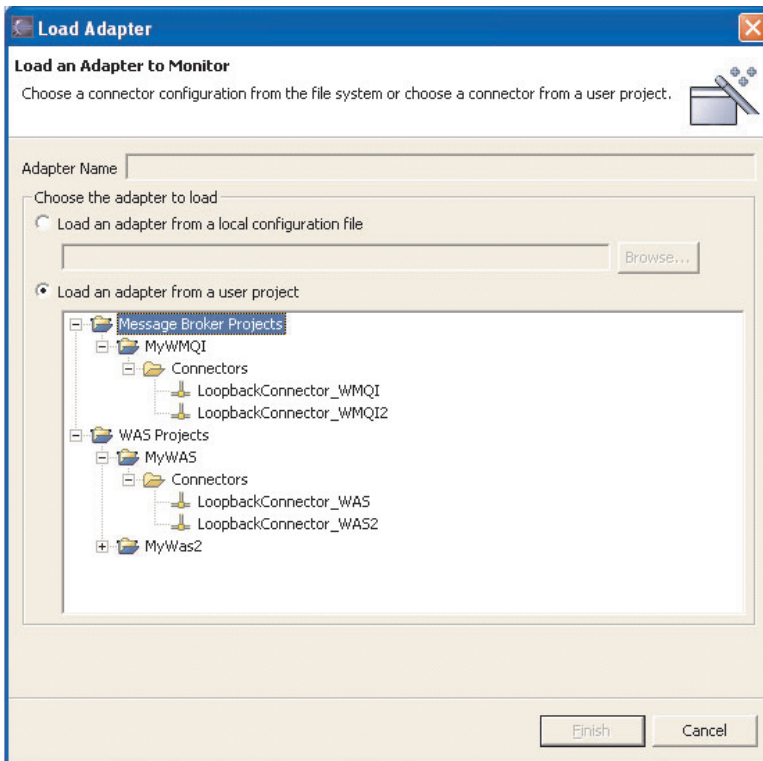


Figure 59. Load Adapter dialog

You can choose to load an adapter either from the adapter’s configuration file or from an integration project.

- If you choose to load from a configuration file, a browse box opens. Navigate to the configuration file for the adapter (typically, the configuration file has a file extension of *.cfg, but other extensions are possible) and choose Save. The name of your selected configuration file appears in the box. Choose Finish. The adapter is loaded into Adapter Monitor and the current data for the adapter is displayed.
- If you choose to load from an integration project, the dialog displays the User Projects currently available in System Manager. Select the connector within a project and click **Finish**. (If no User Projects are available, the display will be empty, and you will need to either create a User Project or load from the configuration file.)

Note: To start the adapter, use the connector startup script, the Start menu shortcut, Visual Test Connector, or another mechanism that you have set up for starting the specific adapter.

Adapter Monitor displays

When you load an adapter, an icon for the adapter appears under the JMS-based Adapters folder in the top left panel. The icon indicates the current state of the

adapter. The act of loading an adapter into Adapter Monitor does not by itself change the state of the adapter. After loading the adapter, you can perform actions to change its state from Adapter Monitor.

The Adapter Monitor displays are refreshed periodically according to the polling time interval that you set for the Adapter Monitor View in its preferences. However, you can perform an immediate refresh at any time by choosing the Refresh views button from the view's toolbar, its drop-down menu, or context menu.

The top left panel of Adapter Monitor is the tree view. It shows JMS-based adapters at the root of the tree and each adapter that has been loaded as tree branches.

The top right panel of Adapter Monitor is the Details panel. The Details panel shows the name of the adapter, its status, and version.

The bottom panel of Adapter Monitor comprises the Fault Queue Manager view. This view shows the messages that have been routed to the queue manager's fault queue. You can use the Fault Queue Manager display to either resubmit or delete messages that were placed in the fault queue as the result of a failed event flow. You can also display additional details of a message in the fault queue by choosing View details.

Changing the state of an adapter

Adapter Monitor enables you to monitor and change the state of an adapter. The state of an adapter refers to the processing that is (or is not) being performed by an adapter.

Note: The existence of an adapter "state" presumes that the adapter has been started. An adapter that has not yet been started, or that was shutdown and not restarted, has no state and is not affected by any actions in Adapter Monitor.

Generally, the adapter performs two types of processing:

- Polling for event notification

The adapter polls the event store of its application for events and sends the events as business object messages to the integration broker.

- Request processing

The adapter receives request business objects sent from the integration broker to the application.

Adapter State	Request Processing	Polling
Active	yes	yes
Paused	yes	no
Inactive	no	no

To change the state of an adapter, right-click on the icon for the adapter, and choose one of the following:

- Activate

Changes the adapter from the Paused or Inactive state to Active

- Deactivate

Changes the adapter from the Paused or Active state to Inactive

- Pause
Changes the adapter from Active state to Paused
- Shutdown
Stops the adapter. This action terminates the connector startup script, closing the connection with the application and freeing any allocated resources. The adapter remains shut down until restarted. The adapter cannot be restarted from Adapter Monitor.

The following two commands do not directly affect the processing of the connector, but do affect what Adapter Monitor displays about the connector:

- Remove from view
Removes this adapter configuration from Adapter Monitor. This does not change the adapter or alter its state.
- Refresh views
Triggers a GETSTATUS command which gets the current status of the connector agent

Using the Fault Queue Manager display

The Fault Queue Manager display shows events that have failed and been received by the fault queue. The display shows as many failed messages as you set in the preferences. The messages are listed in the order in which they were received.

You can select one or more messages to view and then choose View details from either the context menu, the toolbar for the view, or the view's drop-down menu list. Information for that message is displayed in a read-only edit view.

Handling failed events

The Fault Queue Manager lists and enables you to handle the failed event messages from two types of interaction patterns: the HubRequest (synchronous event delivery) interaction pattern and the Agent Delivery (asynchronous event delivery) interaction pattern for container managed events.

You can either delete event messages from the queue or attempt to resubmit them.

You can select one or more events and then choose Delete or Resubmit from either the context menu, the view's toolbar, or the view's drop-down menu.

Adapter Monitor attempts to resubmit the event. If the event is successfully resubmitted, it is removed from the Fault Queue Manager display.

Messages with null value for the ResubmitTo field cannot be resubmitted. An attempt to resubmit an event can fail either because the message itself is invalid, or the ResubmitTo queue is not valid or available.

A message can be invalid because it contains an invalid JMS type or because it cannot be converted to a business object. In either case, when the attempt to resubmit an invalid message fails, Fault Queue Manager displays an error for the invalid message. When you choose OK, the message is not resubmitted and it is removed from the fault queue display. If you close the dialog without choosing OK, the message will remain in the fault queue display.

If a message is valid but the resubmit attempt fails because the message ResubmitTo queue is null, invalid, or unavailable, a Resubmit dialog appears. The

dialog shows the values of the message. You can choose to either retain (by choosing Cancel) or delete (by choosing OK) the message from the queue.

Using Log Viewer to view connector messages

Log Viewer allows you to see messages contained in the log file and the trace file for the connector. You can sort and filter the output display as well as print, save, and email copies of the file. Logging and tracing options, as well as the location of the generated files, are specified as properties in the connector's configuration file.

Note: Log Viewer runs only on a Windows 2000 machine. To configure or view a UNIX log file using Log Viewer, copy the log file from the UNIX machine to a Windows machine and view it from there.

To start Log Viewer, use the Run command from the Start menu and browse for the LogViewer.exe file. Use the Open option of the File menu to browse for the log file. Alternatively, you can create and use a Windows shortcut to LogViewer.exe

Using the Log Viewer menu options, you can perform the following tasks:

- "Setting Log Viewer preferences"
- "Changing how messages are viewed" on page 118
- "Controlling the Log Viewer display output" on page 119

Log Viewer, displaying a sample log file is shown in Figure 60.

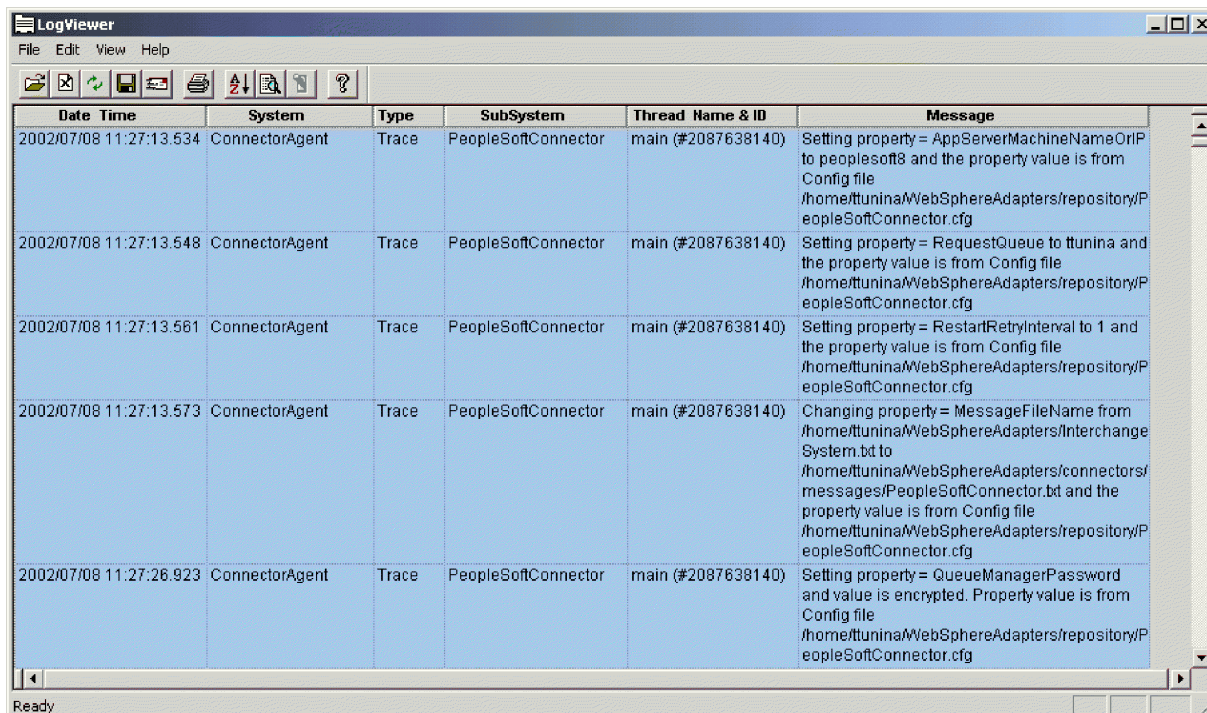


Figure 60. Log Viewer with sample log.

Setting Log Viewer preferences

1. To set Log Viewer preferences, select **Edit > Preferences** from the view's drop-down menu.

The User Configuration Options, General properties dialog box displays (see Figure 61).

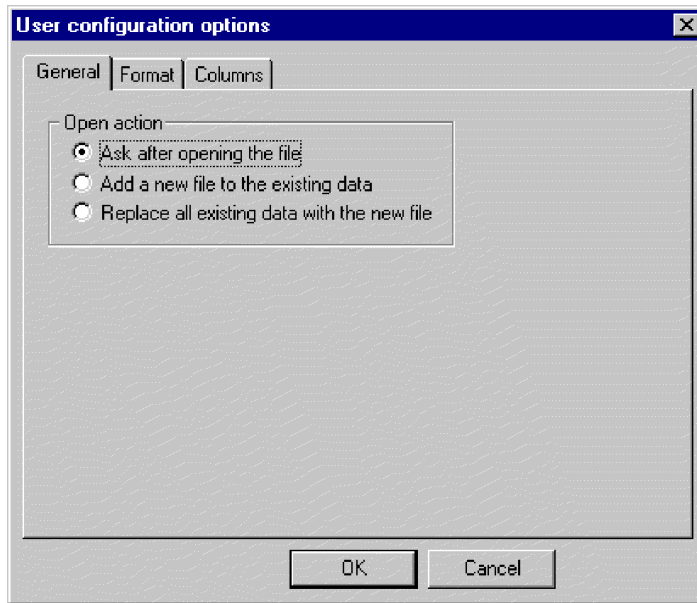


Figure 61. Log Viewer User Configuration Options, General Properties screen.

This dialog lets you specify how to display the log file when you open it. The available choices are:

- Query your preferences each time you open a log file.
 - Merge the log file you are opening with the log file that is currently displayed.
 - Replace the log file that is currently displayed with the contents of the one you are opening.
2. To change the background color and font of the Log Viewer messages, click the **Format** tab.

The User Configuration Options, Format properties dialog box displays (see Figure 62 on page 117).

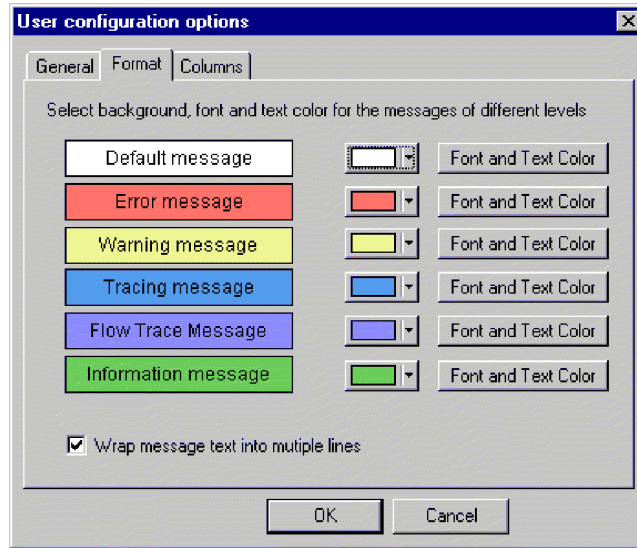


Figure 62. Log Viewer User Configuration Options, Format Properties screen.

This dialog lets you specify how to display the log messages. The available choices are:

- Assign different background colors and fonts for each of the types of messages that display so you can easily recognize their severity (for example, red background with larger font allows for Warning messages).
- Wrap the text of messages if the text is wider than the column.

Note: Flow trace messages are not generated for connectors using WebSphere Application Server.

3. To change the Log Viewer columns that are displayed, click the **Columns** tab. The User Configuration Options, Columns properties dialog box displays (see Figure 63).

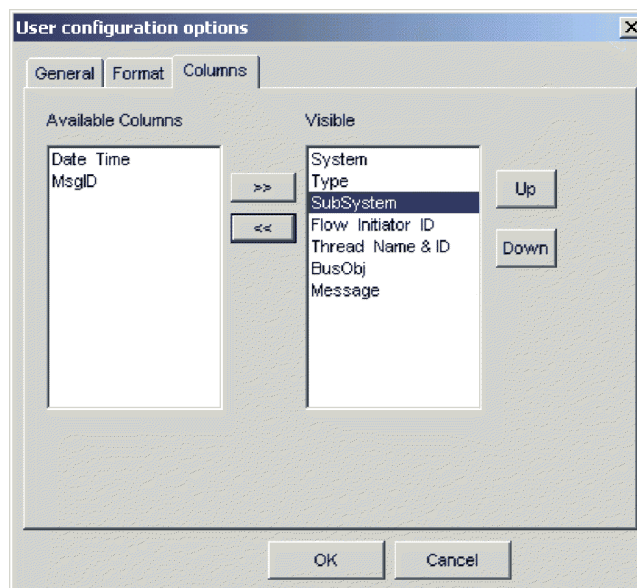


Figure 63. Log Viewer User Configuration Options, Columns Properties screen.

This dialog lets you specify which columns display in Log Viewer as follows:

- To display a column, highlight a column name in the Available Columns pane and click the >> button to move it to the Visible pane.
- To hide a column, highlight a column name in the Visible pane and click the << button to move it to the Available Columns pane.
- Click any of the column names in the Available Columns pane and click the **Up** or **Down** button to change its ordering from left to right in the Log Viewer display. Up moves columns to the left and Down moves columns to the right.
- Click the checkbox next to **Automatically hide empty columns** to keep the Log Viewer display compact.

Note: The column, Flow Initiator ID, is not relevant for connectors using WebSphere Application Server.

Changing how messages are viewed

The View menu contains additional options to change Log Viewer displays. From that menu, you can:

- Display/hide the Log Viewer view's toolbar.
- Display/hide the Log Viewer status bar.
- Split the window into two or more views
- Filter or show all messages by checking filtering options in the filter tabs, such as time range or by type of message (see Figure 64 on page 119 and Table 11 on page 120). To set filter options:
 1. From the view's drop-down menu, select **View > Filter > Use Filter** . The Filter Settings dialog box displays.
 2. In the Activate Filters area, click the box that is associated with the tab containing the filter options you want to apply.
 3. Click **OK** to enable filtering.

The filtered output can be toggled on or off with the Filter Toggle button on the view's toolbar.

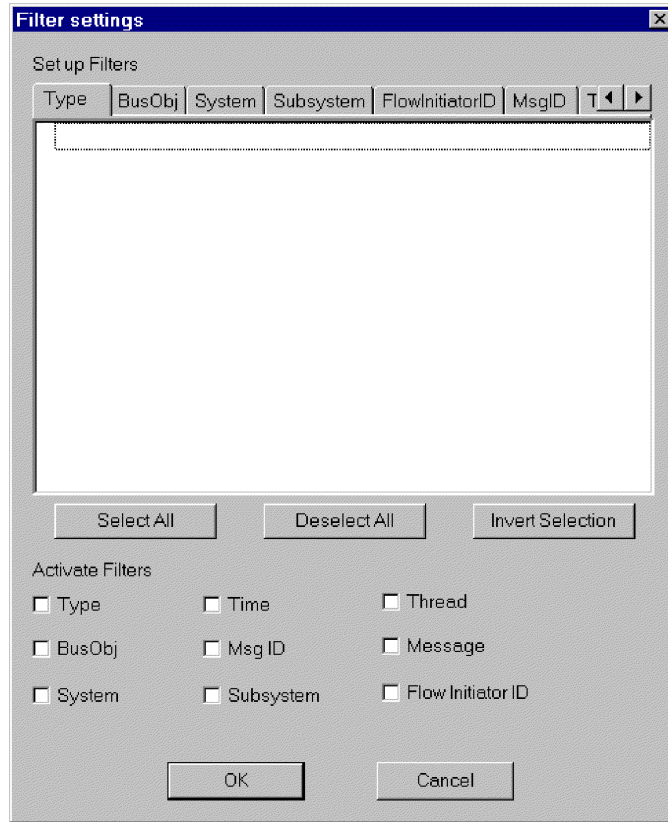


Figure 64. Log Viewer Filter screen.

Note: The tab, FlowInitiator ID, is not relevant for connectors using WebSphere Application Server.

- Sort the messages. Figure 65 shows the Sort options. Click the down arrow in each sort field to select Date/Time or EventID. You can also sort in ascending or descending order.

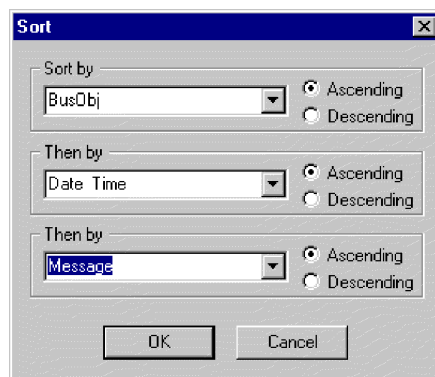


Figure 65. Log Viewer Sort Properties screen.

Controlling the Log Viewer display output

Several options are available for controlling Log Viewer output. The File menu contains options for print previewing, printing, saving, refreshing the display, sending email, and determining the style for page setup, headers, and footers. The variables for header and footers are as follows:

Variable name	Description
\$F	Name of file
\$A	Application name
\$P	Page number
\$N	Total number of pages
\$D	Date (can be followed by additional parameters (for example \$D{%y:%h:%m}))

Filtering messages

To filter the messages that will be displayed in Log Viewer, choose **View->Filter->Use Filter** from the Log Viewer view's drop-down menu. The Filter Settings dialog displays categories that correspond to the parameters of the logging message format. Message format parameters are listed in Table 11.

Table 11. Message format parameters for log file.

Variable	Description
<i>Time</i>	Timestamp: the date of logging in the format <i>year/month/date time</i> .
<i>System</i>	For connectors using the WMQI integration broker, <i>system</i> is the application-specific component of the connector.
<i>Thread</i>	Thread name and thread ID.
<i>Name</i>	The name of the component, such as ClarifyConnector.
<i>MsgType</i>	Indicates the severity of the message. See Table 12 on page 121.
<i>MsgID</i>	The message number.
<i>SubSystem</i>	The connector name.
<i>BO</i>	The business object name.
<i>MsgText</i>	The associated text for the message number.
<i>BOD</i>	Business object dump. The data contained in the business object.

In the Filter Settings dialog you first choose the filtering categories that you want to use. Then you select the specific items that you want to display from each category and choose which filters you want to activate for your current Log Viewer display.

Follow these steps:

1. In the Filter Settings dialog, choose a tab under Set up Filters to display the items that you want to use for filtering messages. For example, choose Time if you want to filter according to the timestamp of the message. You can set up multiple filters and use them either separately or together.
2. In the displayed list of items, select each item for which you want to view messages in Log Viewer. For example, if you want to view only messages that are timestamped between 5 March 2002 at 9:00 AM and 6 March 2002 at 5:00 PM, select the range for those times under the Time tab.

You can use the buttons below the list box to select or deselect all the displayed items or to invert your current selection choices.

3. Under Activate Filters, check the box for each filter type that you want to activate. For example, if you want to see only those messages with a particular message ID that have a particular timestamp, activate both the MsgID filter and the Time filter.
4. Click **OK**. The Filter Settings dialog closes, and the Log Viewer display refreshes to show only those messages that you have allowed through the filters.

Note that in addition to filtering according to the categories, you can also display only those messages that contain a specific text string. To do so, select Messages under Set up Filters, enter the specific text for which you want to show messages, and check the box for **Message under Activate Filters**.

Message types

Table 12 describes the types of messages issued by WebSphere Business Integration adapters.

Table 12. Message types.

Message type	Description
Info	Informational only. You do not need to take action.
Warning	A default condition chosen by InterChange Server.
Error	A serious problem that you should investigate.
Fatal Error	An error that stops operation and should be reported.
Trace	Tracing information for the trace level specified.
Flow Trace	Flow tracing information for business objects.
Internal Error	A serious internal problem that should be investigated.
Internal Fatal Error	An internal error that stops operation. It should be reported.

Note: If a message type of Internal Error or Internal Fatal Error appears, record the circumstances surrounding the problem, and then contact IBM Technical Support.

Appendix A. WebSphere MQ message formats

The following tables list WebSphere MQ message formats, and the settings for particular properties, for the different types of messages exchanged by the connector framework and the integration broker.

Table 13. Format and property settings of event delivery messages from the connector framework to the integration broker.

MQMD	Contains no relevant information.
RFH2 message header	In the <mcd> folder, the following fields contain information that identifies the message, its format, and how it needs to be parsed: <ul style="list-style-type: none">• Message domain is set to mrn. This specifies that the integration broker should use the parser specifically for MRM-managed messages (those that are fully modelled in the message repository).• Message type identifies the name of the highest-level business object represented by the message.• Message set identifies the message set this specific message is associated with. A separate message set is created for each type of business object.• Message format is set to CwXML.
Message body	Contains an XML instance document that conforms to the XML schema and the imported message set definition for the business object specified by the message type in the RFH2 header.

Table 14. Format and property settings of request messages from the connector framework to the integration broker.

MQMD	Reply-to information is located in two fields: ReplyToQ and ReplyToQMgr . They contains the queue name and queue manager name to which the integration broker needs to direct the response message. In JMS messages, these fields specify the JMSReplyTo destination on the request message. MessageID (JMSMessageID) contains a unique value, which is copied to the CorrelID property field on the response message.
RFH2 message header	<p>In the <mcd> folder, the following fields contain information that identifies the message, its format, and how it needs to be parsed:</p> <ul style="list-style-type: none"> • Message domain is set to mrn. This specifies that the integration broker should use the parser specifically for MRM-managed messages (those that are fully modelled in the message repository). • Message type identifies the name of the highest-level business object represented by the message. • Message set identifies the message set this specific message is associated with. A separate message set is created for each type of business object. • Message format is set to CwXML. <p>In the <jms> folder, the Rto (JMSReplyTo) field contains a URI that encodes the queue name and queue manager name to which the integration broker needs to direct the response message. See <i>WebSphere MQ: Using Java</i> for information about how this URI is specified. The reply-to information in Rto and in ReplyToQ/ReplyToQMgr in the MQMD are the same.</p>
Message body	Contains an XML instance document that conforms to the XML schema and the imported message set definition for the business object specified by the message type in the RFH2 header.

Table 15. Format and property settings of response messages from integration broker to the connector framework.

MQMD	The CorrelID field contains the message ID of the request to which the integration broker is responding. For JMS messages, this field is used to define JMSCorrelationID.
RFH2 message header	<p>In the <mcd> folder, the following fields contain information that identifies the message, its format, and how it needs to be parsed:</p> <ul style="list-style-type: none"> • Message domain is set to mrn. This specifies that the integration broker should use the parser specifically for MRM-managed messages (those that are fully modelled in the message repository). • Message type identifies the name of the highest-level business object represented by the message. • Message set identifies the message set this specific message is associated with. A separate message set is created for each type of business object. • Message format is set to CwXML. <p>In the <usr> folder, the following fields contain return status information:</p> <ul style="list-style-type: none"> • Status field contains a string with return status information. Possible string values are: <ul style="list-style-type: none"> -1: The requested operation failed. 0: The requested operation succeeded. 1: The requested operation succeeded. The application has returned a changed business object. • Description field - When status is set to -1, it contains an extended error string with the message sent by the integration broker.
Message body	Contains an XML instance document that conforms to the XML schema and the imported message set definition for the business object specified by the message type in the RFH2 header.

Table 16. Format and property settings for request messages sent from integration broker to the connector framework.

MQMD	<p>Reply-to information is located in two fields: ReplyToQ and ReplyToQMgr. They contains the queue name and queue manager name to which the integration broker needs to direct the response message. In JMS messages, these fields specify the JMSReplyTo destination on the request message. If the ReplyToQ and ReplyToQMgr fields are left blank, the connector framework is not expected to provide a response. If a response is required, messages can also specify reply-to information in the Rto property field of the message header. MessageID (JMSMessageID) contains a unique value, which is copied to the CorrelID property field on the response message.</p>
RFH2 message header	<p>In the <mcd> folder, the following fields contain information that identifies the message, its format, and how it needs to be parsed:</p> <ul style="list-style-type: none"> • Message domain is set to mrm. This specifies that the integration broker should use the parser specifically for MRM-managed messages (those that are fully modelled in the message repository). • Message type identifies the name of the highest-level business object represented by the message. • Message set identifies the message set this specific message is associated with. A separate message set is created for each type of business object. • Message format is set to CwXML. <p>In the <jms> folder, the Rto (JMSReplyTo) property field can optionally contain the queue name and queue manager name to which the connector framework needs to direct the response message.</p>
Message body	<p>Contains an XML instance document that conforms to the XML schema and the imported message set definition for the business object specified by the message type in the RFH2 header.</p>

Table 17. Format and property settings for response messages sent from the connector framework to integration broker.

MQMD	The CorrelID property field contains the message ID of the request to which the connector framework is responding.
RFH2 message header	<p>In the <mcd> folder, the following fields contain information that identifies the message, its format, and how it needs to be parsed:</p> <ul style="list-style-type: none"> • Message domain is set to mrm. This specifies that the integration broker should use the parser specifically for MRM-managed messages (those that are fully modelled in the message repository). • Message type identifies the name of the highest-level business object represented by the message. • Message set identifies the message set this specific message is associated with. A separate message set is created for each type of business object. • Message format is set to CwXML. <p>In the <usr> folder, the following fields contain return status information:</p> <ul style="list-style-type: none"> • Status field contains a string with a return status indicator. Possible string values are: -1: The requested operation failed. 0: The requested operation succeeded. 1: The requested operation succeeded. The application has returned a changed business object. • Description property field - When status is set to -1, it contains an extended error string with the message sent by the connector framework.
Message body	Contains an XML instance document that conforms to the XML schema and the imported message set definition for the business object specified by the message type in the RFH2 header.

Table 18. Format and property settings for administrative messages sent from the connector framework to integration broker.

MQMD	Contains no relevant information.
RFH2 message header	<p>In the <mcd> folder, the following fields contain information that identifies the message, its format, and how it needs to be parsed:</p> <ul style="list-style-type: none"> • Message domain is set to xml to indicate that the message should be parsed by generic XML parser.
Message body	See Appendix C, “Connector startup options,” on page 153 for information about message body content.

Table 19. Format and property settings for administrative messages sent from integration broker to the connector framework

MQMD	If the administrative message is Stop Connector, the Format property is set to: MQC.MQFMT_STRING and the Expiry (JMSExpiration) property field is set to one minute.
Message body	See Appendix C, “Connector startup options,” on page 153 for information about message body content.

Appendix B. Standard configuration properties for connectors

This appendix describes the standard configuration properties for the connector component of WebSphere Business Integration adapters. The information covers connectors running with the following integration brokers:

- WebSphere InterChange Server (ICS)
- WebSphere MQ Integrator, WebSphere MQ Integrator Broker, and WebSphere Business Integration Message Broker, collectively referred to as the WebSphere Message Brokers (and shown as WMQI in the Connector Configurator).
- Information Integrator (II)
- WebSphere Application Server (WAS)

If your adapter supports DB2 Information Integrator, use the WMQI options and the DB2 II standard properties (see the Notes column in Table 20 on page 131.)

The properties you set for the adapter depend on which integration broker you use. You choose the integration broker using Connector Configurator. After you choose the broker, Connector Configurator lists the standard properties you must configure for the adapter.

For information about properties specific to this connector, see the relevant section in this guide.

New properties

These standard properties have been added in this release:

- AdapterHelpName
- BiDi.Application
- BiDi.Broker
- BiDi.Metadata
- BiDi.Transformation
- CommonEventInfrastructure
- CommonEventInfrastructureContextURL
- ControllerEventSequencing
- jms.ListenerConcurrency
- jms.TransportOptimized
- ResultsSetEnabled
- ResultsSetSize
- TivoliTransactionMonitorPerformance

Standard connector properties overview

Connectors have two types of configuration properties:

- Standard configuration properties, which are used by the framework
- Application, or connector-specific, configuration properties, which are used by the agent

These properties determine the adapter framework and the agent run-time behavior.

This section describes how to start Connector Configurator and describes characteristics common to all properties. For information on configuration properties specific to a connector, see its adapter user guide.

Starting Connector Configurator

You configure connector properties from Connector Configurator, which you access from System Manager. For more information on using Connector Configurator, refer to the sections on Connector Configurator in this guide.

Connector Configurator and System Manager run only on the Windows system. If you are running the connector on a UNIX system, you must have a Windows machine with these tools installed.

To set connector properties for a connector that runs on UNIX, you must start up System Manager on the Windows machine, connect to the UNIX integration broker, and bring up Connector Configurator for the connector.

Configuration property values overview

The connector uses the following order to determine a property's value:

1. Default
2. Repository (valid only if WebSphere InterChange Server (ICS) is the integration broker)
3. Local configuration file
4. Command line

The default length of a property field is 255 characters. There is no limit on the length of a STRING property type. The length of an INTEGER type is determined by the server on which the adapter is running.

A connector obtains its configuration values at startup. If you change the value of one or more connector properties during a run-time session, the property's update method determines how the change takes effect.

The update characteristics of a property, that is, how and when a change to the connector properties takes effect, depend on the nature of the property.

There are four update methods for standard connector properties:

- **Dynamic**
The new value takes effect immediately after the change is saved in System Manager. However, if the connector is in stand-alone mode (independently of System Manager), for example, if it is running with one of the WebSphere message brokers, you can change properties only through the configuration file. In this case, a dynamic update is not possible.
- **Agent restart (ICS only)**
The new value takes effect only after you stop and restart the connector agent.
- **Component restart**
The new value takes effect only after the connector is stopped and then restarted in System Manager. You do not need to stop and restart the agent or the server process.

- **System restart**

The new value takes effect only after you stop and restart the connector agent and the server.

To determine how a specific property is updated, refer to the **Update Method** column in the Connector Configurator window, or see the Update Method column in Table 20 on page 131.

There are three locations in which a standard property can reside. Some properties can reside in more than one location.

- **ReposController**

The property resides in the connector controller and is effective only there. If you change the value on the agent side, it does not affect the controller.

- **ReposAgent**

The property resides in the agent and is effective only there. A local configuration can override this value, depending on the property.

- **LocalConfig**

The property resides in the configuration file for the connector and can act only through the configuration file. The controller cannot change the value of the property, and is not aware of changes made to the configuration file unless the system is redeployed to update the controller explicitly.

Standard properties quick-reference

Table 20 provides a quick-reference to the standard connector configuration properties. Not all connectors require all of these properties, and property settings may differ from integration broker to integration broker.

See the section following the table for a description of each property.

Note: In the Notes column in Table 20, the phrase “RepositoryDirectory is set to <REMOTE>” indicates that the broker is InterChange Server. When the broker is WMQI or WAS, the repository directory is set to <ProductDir>\repository

Table 20. Summary of standard configuration properties

Property name	Possible values	Default value	Update method	Notes
AdapterHelpName	One of the valid subdirectories in <ProductDir>\bin\Data\App\Help\ that contains a valid <RegionalSetting> directory	Template name, if valid, or blank field	Component restart	Supported regional settings. Include chs_chn, cht_twn, deu_deu, esn_esp, fra_fra, ita_ita, jpn_jpn, kor_kor, ptb_bra, and enu_usa (default).
AdminInQueue	Valid JMS queue name	<CONNECTORNAME>/ADMININQUEUE	Component restart	This property is valid only when the value of DeliveryTransport is JMS
AdminOutQueue	Valid JMS queue name	<CONNECTORNAME>/ADMINOUTQUEUE	Component restart	This property is valid only when the value of DeliveryTransport is JMS

Table 20. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
AgentConnections	1 through 4	1	Component restart	This property is valid only when the value of DeliveryTransport is MQ or IDL, the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
AgentTraceLevel	0 through 5	0	Dynamic if broker is ICS; otherwise Component restart	
ApplicationName	Application name	The value specified for the connector application name	Component restart	
BiDi.Application	Any valid combination of these bidirectional attributes: 1st letter: I,V 2nd letter: L,R 3rd letter: Y, N 4th letter: S, N 5th letter: H, C, N	ILYNN (five letters)	Component restart	This property is valid only if the value of BiDi.Transforma tion is true
BiDi.Broker	Any valid combination of these bidirectional attributes: 1st letter: I,V 2nd letter: L,R 3rd letter: Y, N 4th letter: S, N 5th letter: H, C, N	ILYNN (five letters)	Component restart	This property is valid only if the value of BiDi.Transformation is true. If the value of BrokerType is ICS, the property is read-only.
BiDi.Metadata	Any valid combination of these bidirectional attributes: 1st letter: I,V 2nd letter: L,R 3rd letter: Y, N 4th letter: S, N 5th letter: H, C, N	ILYNN (five letters)	Component restart	This property is valid only if the value of BiDi.Transformation is true.
BiDi.Transformation	true or false	false	Component restart	This property is valid only if the value of BrokerType is not WAS .
BrokerType	ICS, WMQI, WAS	ICS	Component restart	
CharacterEncoding	Any supported code. The list shows this subset: ascii7, ascii8, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437 .	ascii7	Component restart	This property is valid only for C++ connectors.

Table 20. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
CommonEventInfrastructure	true or false	false	Component restart	
CommonEventInfrastructureURL	A URL string, for example, corbaloc:iiop:host:2809.	No default value.	Component restart	This property is valid only if the value of CommonEventInfrastructure is true.
ConcurrentEventTriggeredFlows	1 through 32,767	1	Component restart	This property is valid only if the value of RepositoryDirectory is set to <REMOTE> and the value of BrokerType is ICS.
ContainerManagedEvents	Blank or JMS	Blank	Component restart	This property is valid only when the value of Delivery Transport is JMS.
ControllerEventSequencing	true or false	true	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
ControllerStoreAndForwardMode	true or false	true	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
ControllerTraceLevel	0 through 5	0	Dynamic	This property is valid only if the value of RepositoryDirectory is set to <REMOTE> and the value of BrokerType is ICS.
DeliveryQueue	Any valid JMS queue name	<CONNECTORNAME>/DELIVERYQUEUE	Component restart	This property is valid only when the value of Delivery Transport is JMS.
DeliveryTransport	MQ, IDL, or JMS	IDL when the value of RepositoryDirectory is <REMOTE>, otherwise JMS	Component restart	If the value of RepositoryDirectory is not <REMOTE>, the only valid value for this property is JMS.
DuplicateEventElimination	true or false	false	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
EnableOidForFlowMonitoring	true or false	false	Component restart	This property is valid only if the value of BrokerType is ICS.
FaultQueue	Any valid queue name.	<CONNECTORNAME>/FAULTQUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
jms.FactoryClassName	CxCommon.Messaging.jms.IBMMQSeriesFactory, CxCommon.Messaging.jms.SonicMQFactory, or any Java class name	CxCommon.Messaging.jms.IBMMQSeriesFactory	Component restart	This property is valid only if the value of DeliveryTransport is JMS.

Table 20. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
jms.ListenerConcurrency	1 through 32767	1	Component restart	This property is valid only if the value of <code>jms.TransportOptimized</code> is true.
jms.MessageBrokerName	If the value of <code>jms.FactoryClassName</code> is IBM, use <code>crossworlds.queue.manager</code> .	<code>crossworlds.queue.manager</code>	Component restart	This property is valid only if the value of <code>DeliveryTransport</code> is JMS.
jms.NumConcurrent Requests	Positive integer	10	Component restart	This property is valid only if the value of <code>DeliveryTransport</code> is JMS.
jms.Password	Any valid password		Component restart	This property is valid only if the value of <code>DeliveryTransport</code> is JMS.
jms.TransportOptimized	true or false	false	Component restart	This property is valid only if the value of <code>DeliveryTransport</code> is JMS and the value of <code>BrokerType</code> is ICS.
jms.UserName	Any valid name		Component restart	This property is valid only if the value of <code>DeliveryTransport</code> is JMS.
JvmMaxHeapSize	Heap size in megabytes	128m	Component restart	This property is valid only if the value of <code>RepositoryDirectory</code> is set to <REMOTE> and the value of <code>BrokerType</code> is ICS.
JvmMaxNativeStackSize	Size of stack in kilobytes	128k	Component restart	This property is valid only if the value of <code>RepositoryDirectory</code> is set to <REMOTE> and the value of <code>BrokerType</code> is ICS.
JvmMinHeapSize	Heap size in megabytes	1m	Component restart	This property is valid only if the value of <code>RepositoryDirectory</code> is set to <REMOTE> and the value of <code>BrokerType</code> is ICS.
ListenerConcurrency	1 through 100	1	Component restart	This property is valid only if the value of <code>DeliveryTransport</code> is MQ.
Locale	This is a subset of the supported locales: en_US, ja_JP, ko_KR, zh_CN, zh_TW, fr_FR, de_DE, it_IT, es_ES, pt_BR	en_US	Component restart	

Table 20. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
LogAtInterchangeEnd	true or false	false	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
MaxEventCapacity	1 through 2147483647	2147483647	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
MessageFileName	Valid file name	InterchangeSystem.txt	Component restart	
MonitorQueue	Any valid queue name	<CONNECTORNAME> /MONITORQUEUE	Component restart	This property is valid only if the value of DuplicateEventElimination is true and ContainerManagedEvents has no value.
OADAutoRestartAgent	true or false	false	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
OADMaxNumRetry	A positive integer	1000	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
OADRetryTimeInterval	A positive integer in minutes	10	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
PollEndTime	HH = 0 through 23 MM = 0 through 59	HH:MM	Component restart	
PollFrequency	A positive integer (in milliseconds)	10000	Dynamic if broker is ICS; otherwise Component restart	
PollQuantity	1 through 500	1	Agent restart	This property is valid only if the value of ContainerManagedEvents is JMS.
PollStartTime	HH = 0 through 23 MM = 0 through 59	HH:MM	Component restart	
RepositoryDirectory	<REMOTE> if the broker is ICS; otherwise any valid local directory.	For ICS, the value is set to <REMOTE> For WMQI and WAS, the value is <ProductDir>\repository	Agent restart	

Table 20. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
RequestQueue	Valid JMS queue name	<CONNECTORNAME>/REQUESTQUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS
ResponseQueue	Valid JMS queue name	<CONNECTORNAME>/RESPONSEQUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
RestartRetryCount	0 through 99	3	Dynamic if ICS; otherwise Component restart	
RestartRetryInterval	A value in minutes from 1 through 2147483647	1	Dynamic if ICS; otherwise Component restart	
ResultSetEnabled	true or false	false	Component restart	Used only by connectors that support DB2II. This property is valid only if the value of DeliveryTransport is JMS, and the value of BrokerType is WMQI.
ResultSetSize	Positive integer	0 (means the results set size is unlimited)	Component restart	Used only by connectors that support DB2II. This property is valid only if the value of ResultSetEnabled is true.
RHF2MessageDomain	mrm or xml	mrm	Component restart	This property is valid only if the value of DeliveryTransport is JMS and the value of WireFormat is CwXML.
SourceQueue	Any valid WebSphere MQ queue name	<CONNECTORNAME>/SOURCEQUEUE	Agent restart	This property is valid only if the value of ContainerManagedEvents is JMS.
SynchronousRequest Queue	Any valid queue name.	<CONNECTORNAME>/SYNCHRONOUSREQUEST QUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
SynchronousRequest Timeout	0 to any number (milliseconds)	0	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
SynchronousResponse Queue	Any valid queue name	<CONNECTORNAME>/SYNCHRONOUSRESPONSE QUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
TivoliMonitorTransaction Performance	true or false	false	Component restart	

Table 20. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
WireFormat	CwXML or CwB0	CwXML	Agent restart	The value of this property must be CwXML if the value of RepositoryDirectory is not set to <REMOTE>. The value must be CwB0 if the value of RepositoryDirectory is set to <REMOTE>.
WsifSynchronousRequest Timeout	0 to any number (milliseconds)	0	Component restart	This property is valid only if the value of BrokerType is WAS.
XMLNamespaceFormat	short or long	short	Agent restart	This property is valid only if the value of BrokerType is WMQI or WAS

Standard properties

This section describes the standard connector configuration properties.

AdapterHelpName

The AdapterHelpName property is the name of a directory in which connector-specific extended help files are located. The directory must be located in <ProductDir>\bin\Data\App\Help and must contain at least the language directory enu_usa. It may contain other directories according to locale.

The default value is the template name if it is valid, or it is blank.

AdminInQueue

The AdminInQueue property specifies the queue that is used by the integration broker to send administrative messages to the connector.

The default value is <CONNECTORNAME>/ADMININQUEUE

AdminOutQueue

The AdminOutQueue property specifies the queue that is used by the connector to send administrative messages to the integration broker.

The default value is <CONNECTORNAME>/ADMINOUTQUEUE

AgentConnections

The AgentConnections property controls the number of ORB (Object Request Broker) connections opened when the ORB initializes.

It is valid only if the value of the RepositoryDirectory is set to <REMOTE> and the value of the DeliveryTransport property is MQ or IDL.

The default value of this property is 1.

AgentTraceLevel

The AgentTraceLevel property sets the level of trace messages for the application-specific component. The connector delivers all trace messages applicable at the tracing level set and lower.

The default value is 0.

ApplicationName

The ApplicationName property uniquely identifies the name of the connector application. This name is used by the system administrator to monitor the integration environment. This property must have a value before you can run the connector.

The default is the name of the connector.

BiDi.Application

The BiDi.Application property specifies the bidirectional format for data coming from an external application into the adapter in the form of any business object supported by this adapter. The property defines the bidirectional attributes of the application data. These attributes are:

- Type of text: implicit or visual (I or V)
- Text direction: left-to-right or right-to-left (L or R)
- Symmetric swapping: on or off (Y or N)
- Shaping (Arabic): on or off (S or N)
- Numerical shaping (Arabic): Hindi, contextual, or nominal (H, C, or N)

This property is valid only if the BiDi.Transformation property value is set to true.

The default value is ILYNN (implicit, left-to-right, on, off, nominal).

BiDi.Broker

The BiDi.Broker property specifies the bidirectional format for data sent from the adapter to the integration broker in the form of any supported business object. It defines the bidirectional attributes of the data, which are as listed under BiDi.Application above.

This property is valid only if the BiDi.Transformation property value is set to true. If the BrokerType property is ICS, the property value is read-only.

The default value is ILYNN (implicit, left-to-right, on, off, nominal).

BiDi.Metadata

The BiDi.Metadata property defines the bidirectional format or attributes for the metadata, which is used by the connector to establish and maintain a link to the external application. The attribute settings are specific to each adapter using the bidirectional capabilities. If your adapter supports bidirectional processing, refer to section on adapter-specific properties for more information.

This property is valid only if the BiDi.Transformation property value is set to true.

The default value is ILYNN (implicit, left-to-right, on, off, nominal).

BiDi.Transformation

The BiDi.Transformation property defines whether the system performs a bidirectional transformation at run time.

If the property value is set to true, the BiDi.Application, BiDi.Broker, and BiDi.Metadata properties are available. If the property value is set to false, they are hidden.

The default value is false.

BrokerType

The BrokerType property identifies the integration broker type that you are using. The possible values are ICS, WMQI (for WMQI, WMQIB or WBIMB), or WAS.

CharacterEncoding

The CharacterEncoding property specifies the character code set used to map from a character (such as a letter of the alphabet, a numeric representation, or a punctuation mark) to a numeric value.

Note: Java-based connectors do not use this property. C++ connectors use the value `ascii7` for this property.

By default, only a subset of supported character encodings is displayed. To add other supported values to the list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory (`<ProductDir>`). For more information, see the Connector Configurator appendix in this guide.

CommonEventInfrastructure

The Common Event Infrastructure (CEI) is a simple event management function handling generated events. The CommonEventInfrastructure property specifies whether the CEI should be invoked at run time.

The default value is false.

CommonEventInfrastructureContextURL

The CommonEventInfrastructureContextURL is used to gain access to the WAS server that executes the Common Event Infrastructure (CEI) server application. This property specifies the URL to be used.

This property is valid only if the value of CommonEventInfrastructure is set to true.

The default value is a blank field.

ConcurrentEventTriggeredFlows

The ConcurrentEventTriggeredFlows property determines how many business objects can be concurrently processed by the connector for event delivery. You set the value of this attribute to the number of business objects that are mapped and delivered concurrently. For example, if you set the value of this property to 5, five business objects are processed concurrently.

Setting this property to a value greater than 1 allows a connector for a source application to map multiple event business objects at the same time and deliver

them to multiple collaboration instances simultaneously. This speeds delivery of business objects to the integration broker, particularly if the business objects use complex maps. Increasing the arrival rate of business objects to collaborations can improve overall performance in the system.

To implement concurrent processing for an entire flow (from a source application to a destination application), the following properties must be configured:

- The collaboration must be configured to use multiple threads by setting its Maximum number of concurrent events property high enough to use multiple threads.
- The destination application's application-specific component must be configured to process requests concurrently. That is, it must be multithreaded, or it must be able to use connector agent parallelism and be configured for multiple processes. The Parallel Process Degree configuration property must be set to a value larger than 1.

The ConcurrentEventTriggeredFlows property has no effect on connector polling, which is single-threaded and is performed serially.

This property is valid only if the value of the RepositoryDirectory property is set to <REMOTE>.

The default value is 1.

ContainerManagedEvents

The ContainerManagedEvents property allows a JMS-enabled connector with a JMS event store to provide guaranteed event delivery, in which an event is removed from the source queue and placed on the destination queue as one JMS transaction.

When this property is set to JMS, the following properties must also be set to enable guaranteed event delivery:

- PollQuantity = 1 to 500
- SourceQueue = /SOURCEQUEUE

You must also configure a data handler with the MimeType and DHClass (data handler class) properties. You can also add DataHandlerConfigMOName (the meta-object name, which is optional). To set those values, use the **Data Handler** tab in Connector Configurator.

Although these properties are adapter-specific, here are some example values:

- MimeType = text/xml
- DHClass = com.crossworlds.DataHandlers.text.xml
- DataHandlerConfigMOName = MO_DataHandler_Default

The fields for these values in the **Data Handler** tab are displayed only if you have set the ContainerManagedEvents property to the value JMS.

Note: When ContainerManagedEvents is set to JMS, the connector does not call its pollForEvents() method, thereby disabling that method's functionality.

The ContainerManagedEvents property is valid only if the value of the DeliveryTransport property is set to JMS.

There is no default value.

ControllerEventSequencing

The ControllerEventSequencing property enables event sequencing in the connector controller.

This property is valid only if the value of the RepositoryDirectory property is set to set to <REMOTE> (BrokerType is ICS).

The default value is true.

ControllerStoreAndForwardMode

The ControllerStoreAndForwardMode property sets the behavior of the connector controller after it detects that the destination application-specific component is unavailable.

If this property is set to true and the destination application-specific component is unavailable when an event reaches ICS, the connector controller blocks the request to the application-specific component. When the application-specific component becomes operational, the controller forwards the request to it.

However, if the destination application's application-specific component becomes unavailable after the connector controller forwards a service call request to it, the connector controller fails the request.

If this property is set to false, the connector controller begins failing all service call requests as soon as it detects that the destination application-specific component is unavailable.

This property is valid only if the value of the RepositoryDirectory property is set to <REMOTE> (the value of the BrokerType property is ICS).

The default value is true.

ControllerTraceLevel

The ControllerTraceLevel property sets the level of trace messages for the connector controller.

This property is valid only if the value of the RepositoryDirectory property is set to set to <REMOTE>.

The default value is 0.

DeliveryQueue

The DeliveryQueue property defines the queue that is used by the connector to send business objects to the integration broker.

This property is valid only if the value of the DeliveryTransport property is set to JMS.

The default value is <CONNECTORNAME>/DELIVERYQUEUE.

DeliveryTransport

The DeliveryTransport property specifies the transport mechanism for the delivery of events. Possible values are MQ for WebSphere MQ, IDL for CORBA IIOP, or JMS for Java Messaging Service.

- If the value of the RepositoryDirectory property is set to <REMOTE>, the value of the DeliveryTransport property can be MQ, IDL, or JMS, and the default is IDL.
- If the value of the RepositoryDirectory property is a local directory, the value can be only JMS.

The connector sends service-call requests and administrative messages over CORBA IIOP if the value of the RepositoryDirectory property is MQ or IDL.

The default value is JMS.

WebSphere MQ and IDL

Use WebSphere MQ rather than IDL for event delivery transport, unless you must have only one product. WebSphere MQ offers the following advantages over IDL:

- Asynchronous communication:
WebSphere MQ allows the application-specific component to poll and persistently store events even when the server is not available.
- Server side performance:
WebSphere MQ provides faster performance on the server side. In optimized mode, WebSphere MQ stores only the pointer to an event in the repository database, while the actual event remains in the WebSphere MQ queue. This prevents writing potentially large events to the repository database.
- Agent side performance:
WebSphere MQ provides faster performance on the application-specific component side. Using WebSphere MQ, the connector polling thread picks up an event, places it in the connector queue, then picks up the next event. This is faster than IDL, which requires the connector polling thread to pick up an event, go across the network into the server process, store the event persistently in the repository database, then pick up the next event.

JMS

The JMS transport mechanism enables communication between the connector and client connector framework using Java Messaging Service (JMS).

If you select JMS as the delivery transport, additional JMS properties such as `jms.MessageBrokerName`, `jms.FactoryClassName`, `jms.Password`, and `jms.UserName` are listed in Connector Configurator. The properties `jms.MessageBrokerName` and `jms.FactoryClassName` are required for this transport.

There may be a memory limitation if you use the JMS transport mechanism for a connector in the following environment:

- AIX 5.0
- WebSphere MQ 5.3.0.1
- ICS is the integration broker

In this environment, you may experience difficulty starting both the connector controller (on the server side) and the connector (on the client side) due to memory use within the WebSphere MQ client. If your installation uses less than 768MB of process heap size, set the following variable and property:

- Set the `LDR_CNTRL` environment variable in the `CWSharedEnv.sh` script.

This script is located in the `\bin` directory below the product directory (`<ProductDir>`). Using a text editor, add the following line as the first line in the `CWSharedEnv.sh` script:

```
export LDR_CNTRL=MAXDATA=0x30000000
```

This line restricts heap memory usage to a maximum of 768 MB (3 segments * 256 MB). If the process memory grows larger than this limit, page swapping can occur, which can adversely affect the performance of your system.

- Set the value of the `IPCCBaseAddress` property to 11 or 12. For more information on this property, see the *System Installation Guide for UNIX*.

DuplicateEventElimination

When the value of this property is true, a JMS-enabled connector can ensure that duplicate events are not delivered to the delivery queue. To use this feature, during connector development, the connector must have a unique event identifier set as the business object `ObjectEventId` attribute in the application-specific code.

Note: When the value of this property is true, the `MonitorQueue` property must be enabled to provide guaranteed event delivery.

The default value is false.

EnableOidForFlowMonitoring

When the value of this property is true, the adapter runtime will mark the incoming `ObjectEventID` as a foreign key for flow monitoring.

This property is only valid if the `BrokerType` property is set to ICS.

The default value is false.

FaultQueue

If the connector experiences an error while processing a message, it moves the message (and a status indicator and description of the problem) to the queue specified in the `FaultQueue` property.

The default value is `<CONNECTORNAME>/FAULTQUEUE`.

jms.FactoryClassName

The `jms.FactoryClassName` property specifies the class name to instantiate for a JMS provider. This property must be set if the value of the `DeliveryTransport` property is JMS.

The default is `CxCommon.Messaging.jms.IBMMQSeriesFactory`.

jms.ListenerConcurrency

The `jms.ListenerConcurrency` property specifies the number of concurrent listeners for the JMS controller. It specifies the number of threads that fetch and process messages concurrently within a controller.

This property is valid only if the value of the `jms.OptimizedTransport` property is true.

The default value is 1.

jms.MessageBrokerName

The `jms.MessageBrokerName` specifies the broker name to use for the JMS provider. You must set this connector property if you specify JMS as the delivery transport mechanism (in the `DeliveryTransport` property).

When you connect to a remote message broker, this property requires the following values:

QueueMgrName:Channel:HostName:PortNumber

where:

QueueMgrName is the name of the queue manager.

Channel is the channel used by the client.

HostName is the name of the machine where the queue manager is to reside.

PortNumber is the port number used by the queue manager for listening

For example:

```
jms.MessageBrokerName = WBIMB.Queue.Manager:CHANNEL1:RemoteMachine:1456
```

The default value is `crossworlds.queue.manager`. Use the default when connecting to a local message broker.

jms.NumConcurrentRequests

The `jms.NumConcurrentRequests` property specifies the maximum number of concurrent service call requests that can be sent to a connector at the same time. Once that maximum is reached, new service calls are blocked and must wait for another request to complete before proceeding.

The default value is 10.

jms.Password

The `jms.Password` property specifies the password for the JMS provider. A value for this property is optional.

There is no default value.

jms.TransportOptimized

The `jms.TransportOptimized` property determines if the WIP (work in progress) is optimized. You must have a WebSphere MQ provider to optimize the WIP. For optimized WIP to operate, the messaging provider must be able to:

1. Read a message without taking it off the queue
2. Delete a message with a specific ID without transferring the entire message to the receiver's memory space
3. Read a message by using a specific ID (needed for recovery purposes)
4. Track the point at which events that have not been read appear.

The JMS APIs cannot be used for optimized WIP because they do not meet conditions 2 and 4 above, but the MQ Java APIs meet all four conditions, and hence are required for optimized WIP.

This property is valid only if the value of `DeliveryTransport` is JMS and the value of `BrokerType` is ICS.

The default value is false.

jms.UserName

the `jms.UserName` property specifies the user name for the JMS provider. A value for this property is optional.

There is no default value.

JvmMaxHeapSize

The `JvmMaxHeapSize` property specifies the maximum heap size for the agent (in megabytes).

This property is valid only if the value for the `RepositoryDirectory` property is set to `<REMOTE>`.

The default value is 128m.

JvmMaxNativeStackSize

The `JvmMaxNativeStackSize` property specifies the maximum native stack size for the agent (in kilobytes).

This property is valid only if the value for the `RepositoryDirectory` property is set to `<REMOTE>`.

The default value is 128k.

JvmMinHeapSize

The `JvmMinHeapSize` property specifies the minimum heap size for the agent (in megabytes).

This property is valid only if the value for the `RepositoryDirectory` property is set to `<REMOTE>`.

The default value is 1m.

ListenerConcurrency

The `ListenerConcurrency` property supports multithreading in WebSphere MQ Listener when ICS is the integration broker. It enables batch writing of multiple events to the database, thereby improving system performance.

This property is valid only with connectors that use MQ transport. The value of the `DeliveryTransport` property must be MQ.

The default value is 1.

Locale

The `Locale` property specifies the language code, country or territory, and, optionally, the associated character code set. The value of this property determines cultural conventions such as collation and sort order of data, date and time formats, and the symbols used in monetary specifications.

A locale name has the following format:

ll_TT.codeset

where:

ll is a two-character language code (in lowercase letters)

TT is a two-letter country or territory code (in uppercase letters)

codeset is the name of the associated character code set (may be optional).

By default, only a subset of supported locales are listed. To add other supported values to the list, you modify the `\Data\Std\stdConnProps.xml` file in the `<ProductDir>\bin` directory. For more information, refer to the Connector Configurator appendix in this guide.

If the connector has not been internationalized, the only valid value for this property is `en_US`. To determine whether a specific connector has been globalized, refer to the user guide for that adapter.

The default value is `en_US`.

LogAtInterchangeEnd

The `LogAtInterchangeEnd` property specifies whether to log errors to the log destination of the integration broker.

Logging to the log destination also turns on e-mail notification, which generates e-mail messages for the recipient specified as the value of `MESSAGE_RECIPIENT` in the `InterchangeSystem.cfg` file when errors or fatal errors occur. For example, when a connector loses its connection to the application, if the value of `LogAtInterChangeEnd` is `true`, an e-mail message is sent to the specified message recipient.

This property is valid only if the value of the `RespositoryDirectory` property is set to `<REMOTE>` (the value of `BrokerType` is `ICS`).

The default value is `false`.

MaxEventCapacity

The `MaxEventCapacity` property specifies maximum number of events in the controller buffer. This property is used by the flow control feature.

This property is valid only if the value of the `RespositoryDirectory` property is set to `<REMOTE>` (the value of `BrokerType` is `ICS`).

The value can be a positive integer between 1 and 2147483647.

The default value is 2147483647.

MessageFileName

The `MessageFileName` property specifies the name of the connector message file. The standard location for the message file is `\connectors\messages` in the product directory. Specify the message file name in an absolute path if the message file is not located in the standard location.

If a connector message file does not exist, the connector uses `InterchangeSystem.txt` as the message file. This file is located in the product directory.

Note: To determine whether a connector has its own message file, see the individual adapter user guide.

The default value is `InterchangeSystem.txt`.

MonitorQueue

The `MonitorQueue` property specifies the logical queue that the connector uses to monitor duplicate events.

It is valid only if the value of the `DeliveryTransport` property is `JMS` and the value of the `DuplicateEventElimination` is `true`.

The default value is `<CONNECTORNAME>/MONITORQUEUE`

OADAutoRestartAgent

the `OADAutoRestartAgent` property specifies whether the connector uses the automatic and remote restart feature. This feature uses the WebSphere MQ-triggered Object Activation Daemon (OAD) to restart the connector after an abnormal shutdown, or to start a remote connector from System Monitor.

This property must be set to `true` to enable the automatic and remote restart feature. For information on how to configure the WebSphere MQ-triggered OAD feature, see the *Installation Guide for Windows* or *for UNIX*.

This property is valid only if the value of the `RespositoryDirectory` property is set to `<REMOTE>` (the value of `BrokerType` is `ICS`).

The default value is `false`.

OADMaxNumRetry

The `OADMaxNumRetry` property specifies the maximum number of times that the WebSphere MQ-triggered Object Activation Daemon (OAD) automatically attempts to restart the connector after an abnormal shutdown. The `OADAutoRestartAgent` property must be set to `true` for this property to take effect.

This property is valid only if the value of the `RespositoryDirectory` property is set to `<REMOTE>` (the value of `BrokerType` is `ICS`).

The default value is `1000`.

OADRetryTimeInterval

The `OADRetryTimeInterval` property specifies the number of minutes in the retry-time interval for the WebSphere MQ-triggered Object Activation Daemon (OAD). If the connector agent does not restart within this retry-time interval, the connector controller asks the OAD to restart the connector agent again. The OAD repeats this retry process as many times as specified by the `OADMaxNumRetry` property. The `OADAutoRestartAgent` property must be set to `true` for this property to take effect.

This property is valid only if the value of the `RespositoryDirectory` property is set to `<REMOTE>` (the value of `BrokerType` is `ICS`).

The default value is `10`.

PollEndTime

The PollEndTime property specifies the time to stop polling the event queue. The format is *HH:MM*, where *HH* is 0 through 23 hours, and *MM* represents 0 through 59 minutes.

You must provide a valid value for this property. The default value is HH:MM without a value, and it must be changed.

If the adapter runtime detects:

- PollStartTime set and PollEndTime not set, or
- PollEndTime set and PollStartTime not set

it will poll using the value configured for the PollFrequency property.

PollFrequency

The PollFrequency property specifies the amount of time (in milliseconds) between the end of one polling action and the start of the next polling action. This is not the interval between polling actions. Rather, the logic is as follows:

- Poll to obtain the number of objects specified by the value of the PollQuantity property.
- Process these objects. For some connectors, this may be partly done on separate threads, which execute asynchronously to the next polling action.
- Delay for the interval specified by the PollFrequency property.
- Repeat the cycle.

The following values are valid for this property:

- The number of milliseconds between polling actions (a positive integer).
- The word *no*, which causes the connector not to poll. Enter the word in lowercase.
- The word *key*, which causes the connector to poll only when you type the letter *p* in the connector Command Prompt window. Enter the word in lowercase.

The default is 10000.

Important: Some connectors have restrictions on the use of this property. Where they exist, these restrictions are documented in the chapter on installing and configuring the adapter.

PollQuantity

The PollQuantity property designates the number of items from the application that the connector polls for. If the adapter has a connector-specific property for setting the poll quantity, the value set in the connector-specific property overrides the standard property value.

This property is valid only if the value of the DeliveryTransport property is JMS, and the ContainerManagedEvents property has a value.

An e-mail message is also considered an event. The connector actions are as follows when it is polled for e-mail.

- When it is polled once, the connector detects the body of the message, which it reads as an attachment. Since no data handler was specified for this mime type, it will then ignore the message.

- The connector processes the first BO attachment. The data handler is available for this MIME type, so it sends the business object to Visual Test Connector.
- When it is polled for the second time, the connector processes the second BO attachment. The data handler is available for this MIME type, so it sends the business object to Visual Test Connector.
- Once it is accepted, the third BO attachment should be transmitted.

PollStartTime

The PollStartTime property specifies the time to start polling the event queue. The format is *HH:MM*, where *HH* is 0 through 23 hours, and *MM* represents 0 through 59 minutes.

You must provide a valid value for this property. The default value is *HH:MM* without a value, and it must be changed.

If the adapter runtime detects:

- PollStartTime set and PollEndTime not set, or
- PollEndTime set and PollStartTime not set

it will poll using the value configured for the PollFrequency property.

RepositoryDirectory

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

If the integration broker is ICS, this value must be set to set to *<REMOTE>* because the connector obtains this information from the InterChange Server repository.

When the integration broker is a WebSphere message broker or WAS, this value is set to *<ProductDir>\repository* by default. However, it may be set to any valid directory name.

RequestQueue

The RequestQueue property specifies the queue that is used by the integration broker to send business objects to the connector.

This property is valid only if the value of the DeliveryTransport property is JMS.

The default value is *<CONNECTORNAME>/REQUESTQUEUE*.

ResponseQueue

The ResponseQueue property specifies the JMS response queue, which delivers a response message from the connector framework to the integration broker. When the integration broker is ICS, the server sends the request and waits for a response message in the JMS response queue.

This property is valid only if the value of the DeliveryTransport property is JMS.

The default value is *<CONNECTORNAME>/RESPONSEQUEUE*.

RestartRetryCount

The RestartRetryCount property specifies the number of times the connector attempts to restart itself. When this property is used for a connector that is connected in parallel, it specifies the number of times the master connector application-specific component attempts to restart the client connector application-specific component.

The default value is 3.

RestartRetryInterval

The RestartRetryInterval property specifies the interval in minutes at which the connector attempts to restart itself. When this property is used for a connector that is linked in parallel, it specifies the interval at which the master connector application-specific component attempts to restart the client connector application-specific component.

Possible values for the property range from 1 through 2147483647.

The default value is 1.

ResultsSetEnabled

The ResultsSetEnabled property enables or disables results set support when Information Integrator is active. This property can be used only if the adapter supports DB2 Information Integrator.

This property is valid only if the value of the DeliveryTransport property is JMS, and the value of BrokerType is WMQI.

The default value is false.

ResultsSetSize

The ResultsSetSize property defines the maximum number of business objects that can be returned to Information Integrator. This property can be used only if the adapter supports DB2 Information Integrator.

This property is valid only if the value of the ResultsSetEnabled property is true.

The default value is 0. This means that the size of the results set is unlimited.

RHF2MessageDomain

The RHF2MessageDomain property allows you to configure the value of the field domain name in the JMS header. When data is sent to a WebSphere message broker over JMS transport, the adapter framework writes JMS header information, with a domain name and a fixed value of mrm. A configurable domain name lets you track how the WebSphere message broker processes the message data.

This is an example header:

```
<mcd><Msd>mrm</Msd><Set>3</Set><Type>  
Retek_POPhyDesc</Type><Fmt>CwXML</Fmt></mcd>
```

This property is valid only if the value of BrokerType is WMQI or WAS. Also, it is valid only if the value of the DeliveryTransport property is JMS, and the value of the WireFormat property is CwXML.

Possible values are `mrm` and `xml`. The default value is `mrm`.

SourceQueue

The `SourceQueue` property designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see “`ContainerManagedEvents`” on page 140.

This property is valid only if the value of `DeliveryTransport` is `JMS`, and a value for `ContainerManagedEvents` is specified.

The default value is `<CONNECTORNAME>/SOURCEQUEUE`.

SynchronousRequestQueue

The `SynchronousRequestQueue` property delivers request messages that require a synchronous response from the connector framework to the broker. This queue is necessary only if the connector uses synchronous execution. With synchronous execution, the connector framework sends a message to the synchronous request queue and waits for a response from the broker on the synchronous response queue. The response message sent to the connector has a correlation ID that matches the ID of the original message.

This property is valid only if the value of `DeliveryTransport` is `JMS`.

The default value is `<CONNECTORNAME>/SYNCHRONOUSREQUESTQUEUE`

SynchronousRequestTimeout

The `SynchronousRequestTimeout` property specifies the time in milliseconds that the connector waits for a response to a synchronous request. If the response is not received within the specified time, the connector moves the original synchronous request message (and error message) to the fault queue.

This property is valid only if the value of `DeliveryTransport` is `JMS`.

The default value is `0`.

SynchronousResponseQueue

The `SynchronousResponseQueue` property delivers response messages in reply to a synchronous request from the broker to the connector framework. This queue is necessary only if the connector uses synchronous execution.

This property is valid only if the value of `DeliveryTransport` is `JMS`.

The default is `<CONNECTORNAME>/SYNCHRONOUSRESPONSEQUEUE`

TivoliMonitorTransactionPerformance

The `TivoliMonitorTransactionPerformance` property specifies whether IBM Tivoli Monitoring for Transaction Performance (ITMTP) is invoked at run time.

The default value is `false`.

WireFormat

The `WireFormat` property specifies the message format on the transport:

- If the value of the RepositoryDirectory property is a local directory, the value is CwXML.
- If the value of the RepositoryDirectory property is a remote directory, the value is CwBO.

WsifSynchronousRequestTimeout

The WsifSynchronousRequestTimeout property specifies the time in milliseconds that the connector waits for a response to a synchronous request. If the response is not received within the specified time, the connector moves the original synchronous request message (and an error message) to the fault queue.

This property is valid only if the value of BrokerType is WAS.

The default value is 0.

XMLNamespaceFormat

The XMLNamespaceFormat property specifies short or long namespaces in the XML format of business object definitions.

This property is valid only if the value of BrokerType is set to WMQI or WAS.

The default value is short.

Appendix C. Connector startup options

The tables below list the options you can specify when starting a connector from Windows or UNIX. Some of these options override selected property settings in the connector's configuration file.

Windows

You can modify the startup for a connector by editing the following to use the connector startup options listed in Table 21:

- The connector's shortcut properties
- The connector's startup file, `start_connName.bat` (for connectors written in Java)
- The connector's startup file, `start_connector.bat` file (for connectors written in C++).

Note: The `-c`, `-n`, and `-s` options are required to start the connector. If you use the shortcut to start the connector, the shortcut properties must include these options in the target field. Similarly, if you start the connector using the `start_connector.bat` or `start_connName.bat`, it must include these options.

Table 21. Connector startup options for Windows.

Option	Description
<code>-c configFile</code>	The full path name of the configuration file to be used during startup. This option is required.
<code>-f pollFrequency</code>	The amount of time between polling actions. Possible values for <code>pollFrequency</code> are: <ul style="list-style-type: none">• The number of milliseconds between polling actions.• <code>key</code>: Causes the connector to poll only when you type the letter <code>p</code> in the connector's Command Prompt window. Enter the word in lowercase.• <code>no</code>: Causes the connector not to poll. Enter the word in lowercase. The default is 10000.
<code>-j</code>	Specifies that the connector is written in Java. This option is optional if you specify <code>-l className</code> .
<code>-l className</code>	Specifies the name of the global class. This option is required for Java connectors.
<code>-n connectorName</code>	Specifies the name of the connector to start. This option is required.
<code>-s brokerName</code>	Specifies the integration broker name. This option is required.
<code>-t</code>	Note: The installer has included (or omitted) the <code>-t</code> option in the shortcut as required for the connector. If you start up a connector from the command line, the value you specify for <code>-t</code> must be the same as the <code>-t</code> value specified in the shortcut. Turns on the connector property, <code>SingleThreadAppCalls</code> . This property guarantees that all calls the connector framework makes to the application-specific connector code are with one event-triggered flow. The default value is <code>false</code> . This property should not be changed from its shipped value. Each connector has the appropriate setting for this option, depending on its architecture.

Table 21. Connector startup options for Windows. (continued)

Option	Description
-x <i>connectorProps</i>	Passes application-specific connector properties to the connector. Use the format <code>prop_name=value</code> for each value you enter.

UNIX

In the UNIX environment, you start a connector by running `connector_manager_connName` script, which is a wrapper for the generic connector manager script (`ProductDir/bin/connector_manager`). The generic connector manager script calls the appropriate `start_connector.sh` script, which handles the actual connector management for the connector.

Each WebSphere Business Integration adapter includes a `start_connector.sh` script. You can modify the `start_connector.sh` script to also include any of the supported startup options listed in Table 22.

Table 22. Options for the `start_connector.sh` script.

Option	Description
-b	This option runs the connector as a background thread; that is, no input is read from STDIN (standard input). The generic <code>connector_manager</code> script (called by each <code>connector_manager_connector</code> script) automatically specifies the <code>-b</code> option when it invokes the <code>start_connector.sh</code> script for a connector. You can remove this option from the script to prevent a connector from being run in the background. The <code>-b</code> option is <i>not</i> valid on the command-line invocation of <code>connector_manager_connector</code> .
-f <i>poll_freq</i>	<p>The amount of time between polling actions. Possible values for <code>poll_freq</code> are:</p> <ul style="list-style-type: none"> • The number of milliseconds between polling actions. • <code>key</code>: Causes the connector to poll only when you type the letter <code>p</code> in the connector's Command Prompt window. Enter the word in lowercase. • <code>no</code>: Causes the connector not to poll. Enter the word in lowercase. <p>The default is 10000. The <code>-f</code> option is valid on the command-line invocation of <code>connector_manager_connector</code>. The connector manager script can pass this option to its associated <code>start_connector.sh</code> script. This option overrides the poll frequency specified in the connector's configuration file.</p>

Table 22. Options for the `start_connector.sh` script. (continued)

Option	Description
<code>-tthreading_type</code>	<p>The <code>threading_type</code> option specifies the threading model.</p> <p>Note: Only use the <code>-t</code> option when you start a custom-developed connector. Connectors that are installed using a pre-supplied IBM adapter installer already have a <code>connector_manager_connector</code> startup script that specifies (or omits) the required <code>-t</code> option, as required by the application, on the line that starts up the connector. Possible values for <code>threading_type</code> are:</p> <ul style="list-style-type: none"> • <code>SINGLE_THREADED</code>: only a single thread accesses the application • <code>MAIN_SINGLE_THREADED</code>: only the main thread accesses the application • <code>MULTI_THREADED</code>: multiple threads can access the application. <p>The <code>-t</code> option is <i>not</i> valid on the command-line invocation of <code>connector_manager_connName</code>. Specify it inside the generic <code>connector_manager</code> script, in the invocation of the <code>start_connector.sh</code> script.</p>

Appendix D. Using the Connector Script Generator tool

The Connector Script Generator utility creates or modifies the connector script for connectors running on the UNIX platform. Use this tool to do either of the following:

- To generate a new connector startup script for a connector you have added without using the WebSphere Business Integration Adapters installer.
- To modify an existing startup script for a connector to include the correct configuration file path.

To run the Connector Script Generator, do the following:

1. Navigate to the *ProductDir/bin* directory.
2. Enter the command `./ConnConfig.sh`.

The Connector Script Generator screen appears as shown in Figure 66.

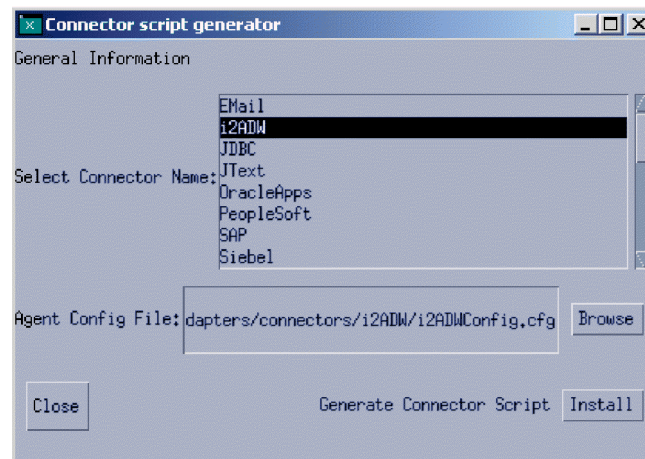


Figure 66. Connector Script Generator.

3. From the Select Connector Name list, select the connector for which the startup script is to be generated.
4. For Agent Config File, specify the connector's configuration file by entering its full-path name or by clicking **Browse** to select a file.
5. To generate or update the connector script, click **Install**.

The connector_manager_ConnectorName file (where *ConnectorName* is the name of the connector you are configuring) is created in the *ProductDir/bin* directory.

6. Click **Close**.

Appendix E. Using Visual Test Connector

Visual Test Connector simulates the activities of a connector to allow you to test your integration components without the complexity of running an actual connector. This chapter consists of the following sections:

- “Recommended testing procedure”
- “Starting Test Connector” on page 160
- “Shutting down Test Connector” on page 160
- “Creating and editing connector profiles” on page 160
- “Emulating a connector” on page 162
- “Working with business objects” on page 162

Recommended testing procedure

This is the recommended test procedure for testing components in the WebSphere business integration system:

1. If your integration broker is InterChange Server, consider using the System View view, which can be very helpful in determining if a flow you have sent ends in success or failure.
For more information, see the *System Administration Guide*.
2. Set up Test Connector to emulate a source connector.
 - a. Launch Test Connector as described in “Starting Test Connector” on page 160.
 - b. Create a profile for the source connector in the interface as described in “Creating a new profile” on page 161.
 - c. Connect Test Connector to the agent to begin emulating the source connector, as described in “Emulating a connector” on page 162.
3. Set up instances of Test Connector to emulate each destination connector involved in the interface.
 - a. Launch Test Connector as described in “Starting Test Connector” on page 160.
 - b. Create a profile for a destination connector as described in “Creating a new profile” on page 161.
 - c. Connect Test Connector to the agent to begin emulating the destination connector as described in “Emulating a connector” on page 162.
 - d. Repeat 3a through 3c above for all destination connectors involved in the interface.
4. Arrange the instances of Test Connector on your screen so that you can easily identify the connector being emulated in each Test Connector window.
5. Send a request business object from the source connector. From the source Test Connector, do the following:
 - a. Create a business object that is managed by the interface you need to test as described in “Creating request business objects” on page 162.
 - b. Save the business object to a file to use in subsequent tests as described in “Saving a business object” on page 165.
 - c. Send the business object as described in “Sending request business objects” on page 163.

6. Simulate the response to the request business object from the destination connector. From the destination Test Connector window, do the following:
 - a. Accept the request business object as described in “Accepting a request business object” on page 166.
 - b. Send the business object as a response as described in “Sending a response business object” on page 167.
7. Repeat step 5 on page 159 through step 6 as many times as necessary to test each interface.

Starting Test Connector

To start Test Connector, do one of the following depending on your integration broker:

- If your integration broker is InterChange Server, select **Start > Programs > IBM WebSphere InterChange Server > IBM WebSphere Business Integration Toolset > Development > Test Connector**.
- If your integration broker is WebSphere Application Server or a WebSphere message broker (WebSphere MQ Integrator, WebSphere MQ IntegratorBroker, or WebSphere Business Integration Message Broker) select **Start > Programs > IBM WebSphere Business Integration Adapters > Tools > Test Connector**.

The Test Connector window includes the following panes:

- The **Business Object Editor** pane in which you can create business object instances to send
- The **Business Object Request List** pane, which displays any business object requests that the connector has received
- The **Output** pane, which displays messages about Test Connector’s operations, such as when a business object has been sent.

Shutting down Test Connector

To shut down Test Connector and cause it to stop emulating a connector agent, select **File > Exit** from the menu bar. When presented with the “Shutdown” prompt, click **Yes**.

Creating and editing connector profiles

Test Connector uses profiles to store the information it needs to emulate a connector. You must create a profile for each connector you want to emulate. You can edit and delete existing profiles.

Saving the connector definition to a file

To emulate a connector using Test Connector, you must save the connector definition to a file. Do the following to save a connector definition to a file:

1. Open the connector definition in Connector Configurator.
2. Select **File > Save As > To File** from the menu bar.
3. Navigate to the directory in which you want the file saved, type a name in the **File name** field, ensure that the value Configuration (*.cfg) is displayed in the Save as type drop-down menu, and click **Save**.

Connector Configurator saves the connector definition to a file with the specified name.

Creating a new profile

You must create a profile for any connector you want to emulate in Test Connector. The profile specifies information such as the name of the connector, the configuration file to be used, and the type of integration broker with which the connector communicates. To create a new connector profile, do the following:

1. Select **File > Create/Select Profile** from the menu bar to display the “Connector Profile” window.
2. In the “Connector Profile window”, select **File > New Profile** from the menu bar.
3. In the “New Profile” window, click **Browse** and then navigate to the configuration file for the connector you preparing in “Saving the connector definition to a file” on page 160.
4. Enter the name of the connector in the **Connector Name** field. You must enter the exact name of the connector definition as it exists in the integration broker repository. For the adapter for JText, for instance, you must type JTextConnector, without any spaces between the words JText and Connector, and with each letter being the proper case.
5. Select the proper integration broker in the **Broker Type** drop-down menu—ICS, WMQI or WAS.

Note: Select WMQI if your broker is any WebSphere message broker.

6. If you selected ICS as your broker type in step 5, do the following as well:
 - a. Type the name of the InterChange Server instance in the **Server** field.
Be sure to type the name precisely; it is case-sensitive and Test Connector will not be able to communicate with InterChange Server if the name is not correct.
 - b. The User ID and Password fields are used and validated only if you are using the ICS Broker Type with Security enabled. Otherwise, these fields are optional.
7. Click **OK** to close the “New Profile” window.

The “Connector Profile” window displays the name of the connector in the **Connector** column, the name of the InterChange Server instance in the **Server** column (if the integration broker is ICS), and the path and name of the connector configuration file in the **Configuration File** column.

8. Click **OK** to close the “Connector Profile” window.

Editing a profile

Follow the steps below to make changes to an existing connector profile:

1. Select **File > Create/Select Profile** from the menu bar of Test Connector or use the keyboard shortcut **Ctrl+N** to display the Connector Profile window.
2. In the “Connector Profile” window select the profile you want to edit and then select **Edit > Edit Profile** from the menu bar.
3. Type new values in the fields of the “New Profile” window and use the **Browse** button to change the configuration file as necessary to make your edits.
4. Click **OK** to close the “New Profile” window.

Deleting a profile

Do the following to delete a connector profile:

1. Select **File > Create/Select Profile** from the menu bar of Test Connector or use the keyboard shortcut **Ctrl+N** to display the “Connector Profile” window.

2. In the “Connector Profile” window, select the profile you want to delete and then select **Edit > Delete Profile** from the menu bar.

Emulating a connector

After creating a profile for a connector, you may use that profile to connect Test Connector to the agent. Once you connect Test Connector to the agent, Test Connector begins emulating the connector defined in the selected profile.

To connect Test Connector to the agent, do the following:

1. Select **File > Create/Select Profile** from the menu bar of Test Connector.
2. In the “Connector Profile” window, select the name of the connector whose profile you want to open.
3. Click **OK**.
4. Select **File > Connect** from the menu bar.

Test Connector displays messages in the “Output” pane as it attempts to emulate the connector. When it finishes connecting, it displays a message indicating that it is ready in the **Output** pane and populates the **Business Object Type** list in the **Business Object Editor** pane.

Working with business objects

To test whether a business process interface has been developed correctly, you need to verify that business objects can be successfully exchanged and processed. This section describes how to:

- Create, modify, delete, and save business object test data
- Compare the attribute values of business objects to easily and quickly view changes made during processing
- Send and receive business objects

Working with request business objects

Request business objects are those that you send from Test Connector when it is emulating a connector that is the source of the events that trigger an interface. Working with request business objects consists of creating a business object instance, populating it with data, and sending the request.

Creating request business objects

To create a new business object in Test Connector, do the following:

1. In the **Business Object Editor** pane, select the name of the business object you want to create from the **Business Object Type** drop-down menu.
2. Click **Create** next to the **Business Object Instance** field.
3. When presented with the **New Instance** dialog, type a name for the instance in the **Enter Name** field.
4. Select the desired verb from the **Verb** drop-down menu.
5. Select the desired locale from the the **Business Object Locale** drop-down menu.
6. Provide values for the simple attributes and child business objects within the top-level object, as described in “Setting values for business object attributes” on page 164.
7. Click **OK**.

Sending request business objects

Once you have created or loaded a business object and specified values for its attributes, you have several ways to send the business object as a request to the integration broker.

Sending request business objects asynchronously: When a source connector sends a request business object in asynchronous mode, it does not expect to get back a response business object. Once the request business object is dispatched, the source connector's role in the transaction is finished. The response business object is typically processed by the integration broker. The default mode for Test Connector is asynchronous.

To send a business object asynchronously, do the following:

1. Select **Request > Mode > Asynchronous** from the menu bar.

Note: Test Connector operates in “Asynchronous” mode by default, so you only have to perform this step if you previously were sending synchronous requests from the connector. Furthermore, you do not have to set the mode before sending each request.

2. Select **Request > Send** from the menu bar.

If the broker specified in the connector definition is InterChange Server then the business object request is sent to the server for processing.

If the broker specified in the connector definition is one of the supported message brokers or WebSphere Application Server then the business object is placed on the queue specified in the RequestQueue standard property.

Sending request business objects synchronously: When a source connector sends a request business object synchronously, it expects to get back a response business object from the integration broker after any destination applications have processed the request. In synchronous mode, Test Connector puts the response business object on the queue specified by the source connector's Synchronous Request Queue property. The default mode for Test Connector is asynchronous.

1. Set Test Connector to synchronous mode by selecting **Request > Mode > Synchronous** from the menu bar.
2. Select **Request > Send** from the menu bar.
3. If the broker specified in the connector definition is InterChange Server then the “Select Collaboration” dialog is displayed. Select the collaboration to which the business object should be sent from the **Collaboration** drop-down menu and click **OK**.

If the broker specified in the connector definition is InterChange Server then the business object request is sent to the configured port of the collaboration object chosen for processing.

If the broker specified in the connector definition is one of the supported message brokers or WebSphere Application Server then the business object is placed on the queue specified in the SynchronousRequestQueue standard property.

Sending request business objects in batch mode: In batch mode, Test Connector lets you specify the number of instances of a particular business object you want to send, as well as one attribute in the top-level object—a primary key attribute, for example—that you want set to a unique value for each instance. Test Connector copies the business object as many times as you have specified, incrementing the

value of the single attribute you specified, and sends each business object. This option allows you to create a large number of business objects quickly and easily.

If the selected attribute is a key field that participates in dynamic cross-referencing as part of an identity relationship, then you must guarantee that the initial value and all those that follow it are unique. Otherwise, the cross-referencing logic will fail, causing the request business objects to fail.

To ensure that the values are unique, you can use Relationship Manager or execute SQL statements against the table for the relationship participant as follows.

- Determine the highest current value for the participant and set the Initial Value field to an even higher value. The first business object instance in the batch and all those that follow will then be unique.
- Delete the existing table entries for the participant, thus guaranteeing that no entries have the same attribute value as any of the batch business objects.

To send business objects in batch mode, do the following:

1. Select the name of the business object you would like to send from the **Business Object Type** drop-down menu.
2. Select **Request > Send Batch** from the menu bar.
3. In the “Batch Mode” window, select the desired verb from the **Verb** drop-down menu.
4. Select the desired locale from the the **Business Object Locale** drop-down menu.
5. Select from the **Attribute** list the attribute in the top-level business object that you want incremented with each business object request in the batch.
The selected attribute should typically be an attribute that uniquely identifies the business object, such as a primary key.
6. In the **Initial Value** field, type the starting value for the attribute to be incremented.
7. In the **No. of Business Objects** field, type the number of business object instances you want generated and sent.
8. Click **OK**.

Test Connector generates the number of business objects you specified, all identical with the exception of the one specified attribute, whose value is incremented for each instance.

If the broker specified in the connector definition is InterChange Server then the business object request is sent to the server for processing.

If the broker specified in the connector definition is one of the supported message brokers or WebSphere Application Server then the business object is placed on the queue specified in the RequestQueue standard property.

Setting values for business object attributes

The following sections describe the various ways you can set the values of simple and compound attributes in a business object instance:

- “Setting values for simple attributes” on page 165
- “Adding child business objects” on page 165
- “Removing child business objects” on page 165
- “Setting the verb of a child business object” on page 165

Setting values for simple attributes

To provide a value for a simple attribute, click its cell in the **Value** column and enter a value.

Adding child business objects

To add an instance of a child business object, right-click the attribute that represents the child object and select **Add Instance** from the context menu.

A plus sign (+) is added next to the attribute that represents child business object to show that there is at least one child business object instance. If you expand the child object attribute, numbered entries are displayed for each instance. The individual instances also have plus signs (+) next to them, so you can expand them and set values for their attributes.

To add more child business object instances, right-click the attribute that represents the child object and select **Add Instance** from the context menu.

Note: If the **Card** property of the attribute that references the child business object is set to the value 1 (indicating it is of single-cardinality), then you will only be able to add one instance of the child object.

Removing child business objects

To remove an instance of a child business object, right-click the instance and select **Remove Instance** from the context menu.

To remove all instances of a child business object, right-click the attribute that represents the child business object and select **Delete All Instances** from the context menu.

Setting the verb of a child business object

You can set the verb of a child business object to test the effect that value has on the business process. This can be helpful when you are troubleshooting logic that involves the cross-referencing of child objects.

To set the verb of a child business object instance, right-click it and choose **Set Verb** from the context menu. When presented with the “Select Verb” prompt, selected the desired verb and click **OK**.

Using the Response BO toolbar

You can edit the attributes of a business object received by a destination connector before you send it as a response. The toolbar of the “Response BO” dialog that you use when doing so has several toolbar buttons that can be used to set the values of the business object. For more information, see “Editing response business objects” on page 167.

Saving a business object

You can save a business object in Test Connector so that it can be used for later tests, shared with technical support (to help troubleshoot problems), or used as response data. You can save any business object, including ones that you have created and ones that appear as requests in the Test Connector window of a destination connector. By default, business objects are saved to a file with a business object extension (.bo).

It is recommended that you create a directory or directory structure specifically for test data files, with subdirectories dedicated to each interface or to each connector, as appropriate. This organization makes the necessary files are easy to locate and

makes testing more efficient. Furthermore, it is recommended that you give the test data file for a business object the same name as the business object definition itself.

Saving request business objects

Do the following to save a business object instance that you have created as a request:

1. Select the business object you want to save.
2. From the menu bar, select **Edit > Save Business Object**.
3. Navigate to the desired directory and specify a name for the file in the **File name** field.
4. Click **Save**.

Saving response business object

Do the following to save a business object instance that has been received by a destination instance of Test Connector and will be sent as a response:

1. Select the business object instance in the “BO Request List” pane.
2. Select **Request > Edit Response** from the menu bar.
3. Click **Save Business Object**.
4. Navigate to the desired directory and specify a name for the file in the **File name** field.
5. Click **Save**.

Loading a business object

To load a business object that has been saved to a file, do the following:

1. Select **Edit > Load Business Object** from the menu bar of Test Connector.
2. Navigate to the business object test data file and open it.
3. When presented with the “New Instance” dialog, type a name for the instance in the **Enter Name** field.
4. Click **OK**.

Deleting a business object

To delete a business object from Test Connector, select **Edit > Delete Business Object** from the menu bar.

Note: This action only removes the business object from the Test Connector. It does not remove the connector’s support for the business object definition.

Accepting a request business object

When you send a business object as a request, the business object appears in the “BO Request List” pane of any Test Connector instances that are emulating destination connectors in the interface, provided that the transaction did not fail.

After you have accepted the request business object, you can edit it if necessary as described in “Editing response business objects” on page 167.

Working with response business objects

Response business objects are those that you send from Test Connector when it is emulating a connector that is the recipient of business object requests in an interface. Working with response business objects consists of editing the values in the business object instance and sending the response back to the broker.

Editing response business objects

When you receive a business object request in a destination instance of Test Connector, you commonly want to edit the values of the attributes. For instance, you will want to provide unique values for primary key attributes that participate in relationships, or you will want to modify the value of other attributes to test map or collaboration logic that responds differently depending on the exact values in the business object.

When it receives a request business object, Test Connector presents the Response Business Object window.

Do the following to set the values of business object attributes :

1. Select the business object instance in the Business Object Editor pane.
2. Select **Request > Edit Response** from the menu bar.
3. Do the following to edit the attributes of the business object:
 - Use one of the techniques described in “Setting values for business object attributes” on page 164 to modify the values of the business object attributes.
 - Click **Reset Business Object to default** to set the values of the business object attributes to their default values as specified in the business object definition.
 - Click **Clear Business Object values** to clear the values of all the attributes in the business object.
 - Click **Load Business Object** to populate the attributes of the business object with test data from a file.

The ability to load saved data into a business object request is very useful in situations where you have to populate a response business object with data before sending it as a reply. Instead of manually typing a value for each attribute that requires response data, you can type the values once, save the business object (as described in “Saving a business object” on page 165), and then load the saved data on subsequent tests.

Sending a response business object

After you accept a request business object, edit the business object, if needed, and send it back as a reply.

Table 23 lists Test Connector’s reply options and shows their corresponding connector return codes for both C++ and Java connectors. For more detailed information about C++ or Java Connector return codes, see the *Connector Development Guide for Java or C++*.

Table 23. Test Connector reply types and connector return codes.

Test Connector reply type	C++ connector return code	Java connector return code
Success	BON_SUCCESS	SUCCESS
Fail	BON_FAIL	FAIL
Multiple Hits	BON_MULTIPLE_HITS	MULTIPLE_HITS
Retrieve By Content Fail	BON_FAIL_RETRIEVE_BY_CONTENT	RETRIEVEBYCONTENT_FAILED
Not Found	BON_BO_DOES_NOT_EXIST	BO_DOES_NOT_EXIST
Value Duplicate	BON_VALDUPES	VALDUPES

To reply to a request business object, do the following:

1. Select the business object in the “BO Request List” pane.
2. From the menu bar, select **Request > Reply**.
3. Select an item from the **Reply** submenu.

Comparing business object instances

Test Connector can compare two business objects of the same type and display the attributes that differ in value. You can use this function to view changes to a business object at different points in the execution of a transaction (for instance, you could compare a business object that has been sent to the integration broker with the same business object after the integration broker has updated it). To compare two business objects, do the following:

1. Create a request business object instance by following the instructions in either “Creating request business objects” on page 162 or “Loading a business object” on page 166.
2. Select the response business object instance in the “BO Request List” pane that you would like to compare the request business object instance to.
3. From the menu bar, select **Edit > Compare Business Objects**.
Test Connector opens the “Compare Business Objects” window with a table that displays the attributes which have different values in the two business objects.
4. Click **OK** to close the window.

Appendix F. Upgrading WebSphere Business Integration adapters

This appendix describes the process for upgrading to new releases of the WebSphere Business Integration Adapter Framework and adapters.

For information on upgrading to Adapter Framework, version 2.6, see *Installing WebSphere Business Integration Adapters, Version 2.6*.

For information on migrating adapters to Adapter Framework, version 2.6, see *Migrating Adapters to Adapter Framework, Version 2.6*.

Index

A

- Adapter Development Kit
 - See* ADK
- Adapter Monitor 110
- adapters 5
- ADK 5
- AdminInQueue 54
- AdminOutQueue 54
- agent delivery 43, 69
- agent request 43, 84
- application client project
 - creating 94
- Application entity 13
- application-specific business object 6
- application-specific business object 16
- application-specific information 19, 21
 - in business object attribute 20
 - in business object verb 31
- application-specific properties (connector) 34
- archiving events 28
- asynchronous data 37
- asynchronous event delivery 69
- attribute (business object) 15, 17, 18
 - application-specific information for 20
 - data types of 18
- Available Integration Component Libraries 67

B

- basic attribute type 18
- bindings mode 56
- business object
 - adding the initialization library 101
 - application-specific 16, 19, 21
 - application-specific information in 19
 - as event notification 13
 - as request 13
 - as response 13
 - attribute 15, 17, 18
 - attribute values 15
 - child 15
 - components 14
 - construction 31
 - creating definitions 52
 - deconstruction 31
 - definitions 6
 - designating supported 59
 - flat 15
 - generic 16
 - hierarchical 15
 - requirement for initiating interaction patterns 101
 - roles of 13, 14
 - type 14
 - verb 14
- Business object
 - mapping viii

- business object definition
 - connector download of 23
- business objects
 - application-specific 6

C

- child business object 15
- Client mode 56
- compound attribute type 18
- configuration
 - connector 34
 - connector logging 62
 - connector startup files, shortcuts, and environment variables 63
 - defining queue 56
 - saving as ICLs 65
 - the connector 57
 - tracing 62
 - WebSphere queues 53
- configuration tasks 51
- connector 5
 - configuration 23, 34
 - creating multiple instances of 107
 - development 34
 - event notification behavior 23, 29
 - modification 34
 - properties 34
 - request processing behavior 34
 - starting 105
 - startup options 153
 - stopping 106
 - tracing 63
- connector agent
 - constructing business objects 31, 33
 - processing events 27, 29
 - processing requests 29, 34
- Connector Configurator
 - running 58
 - using 53
- Connector Script Generator 157
- constructing business objects 31, 33
- correlation ID 10

D

- data handler 8
- data type (attribute) 18
- deconstructing business objects 31, 33
- DeliveryQueue 54
- deploying user projects 67

E

- e-business
 - integrating applications 4
- EJB
 - creating 72, 91
- EJB project
 - creating 85, 99

- environment variables 63
- Error message
 - severity 121
 - type 121
- event
 - archiving 28
 - inbox 25
 - text of 28
- event delivery 7, 38
- event notification 7, 23, 29
 - business object role in 13
 - connector role in 24
 - setting up 24
- event store 7

F

- Fault Queue 110
- FaultQueue 54
- flat business object 15

G

- generic business object 16

H

- hierarchical business object 15
- HP operating system 48
- hub one way 43, 90
- hub request 43, 98

I

- IBM WebSphere MQ
 - installing 46
- IBM WebSphere MQ, definition 48
- installing
 - IBM WebSphere MQ 46
 - JDK (Java Development Kit) 46, 48
 - WebSphere MQ 48
- integration broker 4, 37

J

- java beans 68
- Java compiler.
 - See* JDK
- Java Database Connectivity.
 - See* JDBC
- Java Development Kit.
 - See* JDK
- JDK (Java Development Kit) 46
 - installing 46, 48
- JMS 5

L

- listening mechanism 9
- log and trace files
 - archiving of 109
 - managing 108
- logging 62
- LogViewer
 - using 115

M

- MDB
 - creating 86
- MDB (message driver bean)
 - creating 77
- message body 39
- message descriptor 38
- message formats 38
- message header 39
- Messages (CrossWorlds)
 - format 121
- metadata 6

O

- Object Discovery Agent
 - See ODA
- Object Discovery Agent Development Kit
 - See ODK
- ODA 5
 - using 52
- ODK 5

P

- poll call 7
- polling
 - by connector agent 27
 - configurable properties 27
- properties
 - connector 34
 - setting standard and connector-specific 59
- Publish and subscribe viii

Q

- queue
 - WebSphere MQ 8
- queue manager 39
 - setting the connection mode 60
- queues 39
 - binding mode 56
 - client mode 56
 - configuration 56
 - specifying for use by the connector 60

R

- request 9
- request business object 13
- request processing 10, 34, 38
- request processing (connector) 31

- RequestQueue 54
- requirements
 - software 45, 47
 - web browser 46, 48
- response 10
- response business object 13
- response processing 11
- ResponseQueue 54

S

- Severity (of messages) 121
- shortcuts 63
- software requirements 45, 47
- Solaris operating system 48
- standard properties (connector) 34
- starting a connector 105
- startup files 63
- startup options 153
- stopping a connector 106
- synchronous data transport 37
- synchronous event delivery 84
- SynchronousRequestQueue 54
- SynchronousResponseQueue 54
- System Manager
 - creating user projects in 67

T

- testing 94
- trace levels 62
- tracing 62
 - connector 63
 - difference from logging 61

U

- user projects in System Manager 67
 - deploying 67

V

- verb 14, 18
 - application-specific information for 21, 31
- Visual Test Connector 64, 159

W

- web browser requirements 46, 48
- WebSphere Application Server
 - interaction patterns 43
- WebSphere Application Server applications
 - creating 68
- WebSphere business integration system 4
- WebSphere MQ 8
 - installing 48
- WebSphere MQ message formats 123
- WebSphere MQ queues 39
 - clearing messages from 108
 - configuring for WAS 53
 - creating 54
- Windows 2000 46

- workflow (notification example) 25

Notices

IBM may not offer the products, services, or features discussed in this document in all 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 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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Burlingame Laboratory Director
IBM Burlingame Laboratory
577 Airport Blvd., Suite 800

Burlingame, CA 94010
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not necessarily 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 may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may 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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

COPYRIGHT LICENSE

This information may contain 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.

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change. System Manager and other perspectives include software developed by the Eclipse Project (<http://www.eclipse.org/>)



IBM WebSphere Business Integration Adapter Framework V2.6.0.



Printed in USA