WebSphere® Partner Agreement Manager

Adapters for MQSeries User's Guide

Version 2 Release 1

BIAAAG00

Note: Before using this information and the product it supports, read the information in *Notices* on page 117.

Second Edition (April 2001)

This edition applies to version 2, release 1 of WebSphere Partner Agreement Manager (product number 5724-A85) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. You can make comments on this information via e-mail at idrcf@hursley.ibm.com.

Table of Contents

Welcome to Adapters for MQSeries User's Guide

This document describes the WebSphere® Partner Agreement Manager Adapters for MQSeries® and explains how to configure, customize, debug, and use the Adapters for MQSeries.

To customize your own adapter(s), see these chapters:

- *Introducing Adapters for MQSeries* on page 1 provides an overview of IBM MQSeries and how it relates to Adapters for MQSeries.
- *Installing Adapters for MQSeries* on page 5 provides general installation information that applies to all the adapters.
- *Using Adapter for MQSeries Messaging* on page 9 explains how to customize and use the basic messaging adapter.
- *Using Adapter for MQSeries Integrator (RFH)* on page 29 explains how to customize and use the integrator adapter.
- *Using Adapter for MQSeries Integrator (RFH2)* on page 49 explains how to customize and use the latest integrator adapter.
- *Using Adapter for MQSeries Publish/Subscribe* on page 81 explains how to customize and use the publish/subscribe adapter.
- *Using Adapter for MQSeries Workflow* on page 101 explains how to customize and use the workflow adapter.

Who should use this information

This information is for those who need to create their own customized Adapters for MQSeries.

Related information

For additional information see the following:

- The readme.txt file. This file may contain information that became available after this book was published. Before installation, the readme.txt file is located in the root directory of the product CD-ROM. After installation, the readme.txt file is located in the root directory of the Adapters for MQSeries installation.
- The *Partner Agreement Manager Installation Guide*, form number GC34-5964-00, which describes how to install Partner Agreement Manager.
- The *Partner Agreement Manager Administrator's Guide*, form number BIAAAB00, which describes how to set up, configure, and administer Partner Agreement Manager after you install it.
- The *Partner Agreement Manager User's Guide*, form number BIAAAC00, which describes how to start a Partner Agreement Manager session, design public and private processes, define element definition sets, create business objects, and manage process distribution.
- The *Partner Agreement Manager Adapter Developer's Guide*, form number BIAAAD00, which describes how to develop and administer adapters using the Partner Agreement Manager Adapter Development Environment.
- The *Partner Agreement Manager Script Developer's Guide*, form number BIAAAE00, which describes how to write scripts used in Partner Agreement Manager private processes and elsewhere.
- The *Partner Agreement Manager API Guide*, form number BIAAAF00, which describes principles behind the Partner Agreement Manager External API. See also, the Javadocs for the External API, which you can access from the *Partner Agreement Manager API Guide*.
- The *Partner Agreement View User's Guide*, form number GC34-5965-00, which describes how to install, configure, and use Partner Agreement View.9b1l

Summary of Changes

This edition includes these changes since the previous, first, edition:

- *External APIs.* Partner Agreement Manager 2.1 provides added flexibility to external applications through additional APIs. These APIs allow third-party applications to take advantage of the Partner Agreement Manager partner management and process engine through programmatic access. The API is distributed as a set of Java classes that the external application can import. Communication between the API classes and the Process Server is through RMI, but in the future can be swapped out for HTTP or SOAP. Specifically, APIs have been added to the following functional areas:
  - Session Service API
  - Admin Service API
  - Document Service API
  - Partner Service API
  - Adapter Service API
  - Process Service API

- *LDAP Support.* Partner Agreement Manager 2.1 provides centralized user authentication and administration through an LDAP directory. Partner Agreement Manager can retrieve user information—such as name, e-mail address, phone, and fax—stored in an LDAP directory. Updating this information is done in a single place, through the LDAP management tool. Users are authenticated through the same directory, giving them single-sign-on capabilities across enterprise applications.
- *Double-byte character sets (DBCS) and National Language Support (NLS).* Double-byte character sets are now supported in Partner Agreement Manager 2.1. Double-byte and multibyte data can be transferred and operated on in business objects and adapters. NLS lets Partner Agreement Manager display user interface text in other languages.
- *Improved XML Support.* The Partner Agreement Manager 2.1 engine fundamentally changes the way it interacts with business objects by replacing proprietary parsers with a third-party parser. This simplifies support of DTD 1.0 and the support of XML schemas when the standard is finalized.

  The Business Object and Script API have been extended with new classes and methods. The new classes and methods let you work with business objects as W3C Documents.
- *Adapter Asynchronous Callback.* An additional Adapter API allows adapters to be more efficient with long-running adapter operations. The Asynchronous Callback method tells the Adapter Server that an operation will be long-running, that system resources should be freed while the adapter waits for a response from the end system, and that another method will be called when the response arrives. The Asynchronous Callback method frees the adapter developer from using the request-retry method that makes the Adapter Server responsible for polling the end system for the response.
- *Script API Changes.* The script API now provides access to the PartnerGroupContext and the Public and Private Process Contexts. Through these contexts, you can get information such as partner group binding, a reference to the process, inputs to the process (which contain a reference to the sender and, the ID of the sending node, and the variable name), and unique node and loop IDs.
- *Certificate Support.* Partner Agreement Manager 2.1 is able to request and import certificates from certificate authorities like VeriSign. This lets organizations use their existing certificate, or request a new one if their partners do not accept self-signed certificates.

- *Outbound Proxy Support.* Partner Agreement Manager 2.1 channels that use HTTP communication can work with outbound proxies that use authentication. Outbound proxy authentication is used within *internal* networks to ensure that only people and applications that are authenticated may communicate with an *external* network. Authentication in the outbound proxy is done with a standard username and password combination. You can turn on the outbound proxy feature after installation. Thereafter, all outbound HTTP communication will use the same username and password combination for the proxy.

NOTE: Note that this feature is only used by channels using HTTP communication; it does not apply to channels that use the built-in Partner Agreement Manager proxy.

CHAPTER

1

INTRODUCING ADAPTERS FOR MQSERIES

Welcome to WebSphere® Partner Agreement Manager Adapters for MQSeries®, the business-to-business integration system tailored to work specifically with IBM MQSeries.

This chapter includes the following sections:

- *Introducing MQSeries* on page 2.
- *Introducing Adapters for MQSeries* on page 3.

The WebSphere Partner Agreement Manager XML-based e-commerce solution allows Global 2000 companies to dramatically improve both their efficiencies and competitiveness by leveraging the Internet to automate critical business processes and the flow of information between partners, customers, and suppliers.

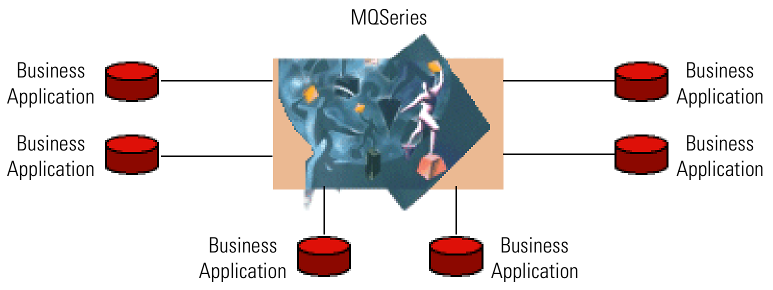
Adapters for MQSeries combine the flexibility of MQSeries with the efficiency of Partner Agreement Manager (PAM), providing the infrastructure required to automate complete business processes with both internal and external interfaces.

Note: Partner Agreement Connect (PAC) is a version of Partner Agreement Manager (PAM) designed for partners who do not initiate processes. Because the same core documentation set is used for both PAM and PAC, the documentation uses the term "Partner Agreement Manager" or "PAM" to refer to both products.

Introducing MQSeries

IBM MQSeries, the most widely used message-queueing software on the market, enables users to exchange information between applications across more than 35 different platforms, from mainframes to PCs.

IBM's MQSeries is a key element of enterprise systems integration—an open, scalable messaging and information infrastructure—that provides any-to-any connectivity across platforms, from desktop PCs to mainframes.



MQSeries is a queue-based messaging tool that stores and forwards messages asynchronously. Messages consist of the content plus specialized descriptive fields (known collectively as the MQ *message descriptor*). No metadata or self-descriptive information is inherent in the message content, and no restrictions are placed on how data is represented. All meaning is at the application level, and understanding of message formats is usually hard-coded into the applications themselves. Adapter for MQSeries Messaging is an example of such an application.

Key MQSeries features include:

- point-to-point message communications.
- MQSeries messages are sent and received via queues.
- asynchronous transit.
- synchronous client access.

- multiple queue types, including local queues, remote queues, alias queues, transmission queues, and dead-letter queues.
- dynamic configuration and administration.
- segmentation of large messages.
- support for binary messages.
- failure recovery.
- acknowledgment/Reply messages.

For more information about MQSeries, visit the IBM web site at http://www.ibm.com/software/ts/MQSeries/.

Introducing Adapters for MQSeries

One of the primary objectives of Partner Agreement Manager is to allow companies to take full advantage of their existing infrastructure. This infrastructure comprises not only software and hardware, but also the employee skill sets and the operations infrastructure of a company. The tight interface with MQSeries allows Adapters for MQSeries both to communicate with applications that are already MQSeries-enabled and to continue to leverage existing methodologies and knowledge bases when establishing connections with additional applications. Adapters for MQSeries provide all the components required for integration with MQSeries.

About Adapters for MQSeries

In addition to the core Partner Agreement Manager product, Adapters for MQSeries communicate with internal applications, which handle connections, the exchange of business objects, transaction contexts, and data transformation.

Adapters for MQSeries include:

- Adapter for MQSeries Messaging
- Adapter for MQSeries Integration (RFH support)
- Adapter for MQSeries Integration (RFH2 support)
- Adapter for MQSeries Publish/Subscribe
- Adapter for MQSeries Workflow

Adapter for MQSeries Messaging

Adapter for MQSeries Messaging works with the base messaging features of MQSeries. MQSeries Messaging is the core of the IBM MQSeries family, providing once-only message and queueing capabilities on over 35 platforms.

See *Using Adapter for MQSeries Messaging* for more information.

Adapter for MQSeries Integrator (RFH)

Adapter for MQSeries Integrator (RFH) facilitates the connection of external partners to information flows that have been established with the message brokering capabilities of MQSeries Integrator, using the MQSeries Rules and Format Header (MQRFH). The product manages the secure and reliable communication of messages as part of intercompany processes.

See *Using Adapter for MQSeries Integrator (RFH)* for more information.

Adapter for MQSeries Integrator (RFH2)

Adapter for MQSeries Integrator (RFH2) facilitates the connection of external partners to information flows that have been established with the message brokering capabilities of MQSeries Integrator, using the updated MQSeries Rules and Format Header (MQRFH2).

See *Using Adapter for MQSeries Integrator (RFH2)* for more information.

Adapter for MQSeries Publish/Subscribe

Adapter for MQSeries Publish/Subscribe provides access to MQSeries Publish/Subscribe message brokering capabilities.

See *Using Adapter for MQSeries Publish/Subscribe* for more information.

Adapter for MQSeries Workflow

Adapter for MQSeries Workflow allows internal workflow steps that have been automated with MQSeries Workflow to be incorporated into larger inter-company processes. Conversely, it also allows inter-company exchanges to be incorporated into internal workflows.

See *Using Adapter for MQSeries Workflow* for more information.

CHAPTER 2

INSTALLING ADAPTERS FOR MQSERIES

Read this chapter to learn how to install the WebSphere® Partner Agreement Manager Adapters for MQSeries® version 2.1.

This chapter includes the following sections:

- *System requirements* on page 6.
- *Server hardware requirements on Windows NT* on page 6.
- *Software requirements on Windows NT* on page 7.
- *Importing key elements* on page 7.

Specific configuration information can be found in the subsequent chapters for each individual adapter.

NOTE: Partner Agreement Connect (PAC) is a version of Partner Agreement Manager (PAM) designed for partners who do not initiate processes. Because the same core documentation set is used for both PAM and PAC, the documentation uses the term “Partner Agreement Manager” or “PAM” to refer to both products.

System requirements

Warning: Make sure you *back up* your partner root directory *before* installing. This is especially important if you have ever used (and customized) previous versions of MQSeries adapters.

This section describes the products and versions required. These are the current system requirements for Adapters for MQSeries.

Tip: For best adapter performance, be sure that the computer your Adapter Server is running on meets the minimum Partner Agreement Manager system requirements. See the *Partner Agreement Manager Installation Guide.*

Server hardware requirements on Windows NT

- Pentium II 300 MHz processor (or better)
- 1 MB or higher L2 cache
- 128 MB RAM (or more)
- Although you can run the Partner Agreement Manager server with only 64 MB RAM, you achieve significantly better performance with 128 MB RAM.
- 4 GB hard drive (or larger)

  The hard disk requirement for the Partner Agreement Manager server and all supporting software is approximately 390 MB. Make provisions for a 1 GB (or more) database device and a 250 MB (or more) database log device.

Note: Plan to allocate additional disk space to archive audit information, because the total amount of disk space you actually need is variable. The size of the business objects that pass between you and your PAM partners also affects the amount of disk space required. If you expect to create business objects that contain complicated schematics or CAD drawings, for example, plan to allocate significantly more disk space.

Software requirements on Windows NT

- Windows NT Server 4.0 or later operating system with Service Pack (SP) 3 or later

Important: Be sure to configure the NT Server used to host Partner Agreement Manager to use the NTFS file system rather than the FAT (file allocation table) file system. Using FAT disables many of Windows NT's built-in security features.

- IBM DB2® version 7.1 (see http://www-4.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/download.d2w/report) with FixPak 2
- JDBC driver 2
- Partner Agreement Manager or Partner Agreement Connect version 2.1
- IBM HTTP Server 1.3.12.2
- Internet Explorer 4.0 or later (available from www.microsoft.com/msdownload)
- Network access to an SMTP server (for e-mail notifications), plus an e-mail account that PAM can use for sending e-mail for notifications
- A certificate for the web server (available from any certificate authority, such as www.verisign.com, depending on the local security policy)

Importing key elements

This section describes importing business objects, adapter types and implementations, and creating an instance.

Note: You must move all the XML files for business object exports and adapter types/implementations from the Partner Agreement Manager server computer to the computer running the PAM client. You can do this using any of the following methods:

- Use File Transfer Protocol (FTP) to transfer the files from the PAM server to the local PAM client computer. Make sure you use binary mode to FTP the files.

- Use file sharing software to make the partner directory remotely accessible across your network so the PAM client can directly access the XML files to import them.
- Load the zip file (NT) from the CD onto the local PAM client computer and manually extract the files.

WARNING: When you start customizing an adapter, it is a good idea to do so in your own work space, for example:
<partner_root>\com\<**your_company**>\adapters\ibm\mqseries\test
This helps ensure that any future Adapters for MQSeries updates do not overwrite your custom work.

CHAPTER
3

USING ADAPTER FOR MQSERIES MESSAGING

Read this chapter to learn how to configure and use WebSphere® Partner Agreement Manager Adapter for MQSeries® Messaging version 2.1.

This chapter includes the following sections:

Adapter for MQSeries Messaging combines the messaging and queueing features of IBM MQSeries Messaging with the business-to-business integration capabilities of WebSphere Partner Agreement Manager.

NOTE: Partner Agreement Connect (PAC) is a version of Partner Agreement Manager (PAM) designed for partners who do not initiate processes. Because the same core documentation set is used for both PAM and PAC, the documentation uses the term “Partner Agreement Manager” or “PAM” to refer to both products.

ABOUT ADAPTER FOR MQSERIES MESSAGING

This version of Adapter for MQSeries Messaging is compatible with WebSphere Partner Agreement Manager (PAM) version 2.1. It requires the IBM MQSeries classes for Java (MQ base Java)™, which are compatible with all versions of MQSeries through version 5.1.

Adapter for MQSeries Messaging consists of:

- Class files corresponding to all Java classes.
- Java files for MQSeriesAdapter and MQSeriesAdapterCustom classes.
- Java and class files for an example MQSeriesAdapterCustom subclass (MQSeriesCustomExample.java and MQSeriesCustomExample.class).
- XML representations of template adapter type and implementation (MQSeriesAdapterType.xml, MQSeriesAdapterJavaImp.xml).
- Javadoc reference for all class files.

Key features of Adapter for MQSeries Messaging include:

- *The ability to perform request-reply operations* with other applications on the MQSeries network, in addition to just reading a message from a queue or putting a message on a queue. You should bear in mind the distinction between just reading/writing messages to/from queues versus actually communicating with endpoint applications. The latter usually requires some kind of two-way request-reply communication, especially if you are expecting some kind of response from the application. For example, from the perspective of a PAM process designer, a simple Post operation is executed against a particular end system. But, from your perspective, Adapter for MQSeries Messaging is sending a message to a queue, and might need to wait for an acknowledgment.
- *On-site customizability* of data transformation and reply behavior via custom methods—per adapter instance.
- *Three data transformation* options that allow you to:

    - Call out to third-party mapping tools that handle the mapping between MQSeries message contents and XML corresponding to PAM business objects. These maps must be set up separately as command-line executables or scripts that can be executed from the adapter. As long as the XML written by or read into the map matches the given PAM business object, you can implement full data transformation.
    - Custom-code mapping in custom Java methods, to be called at the appropriate times by the adapter's core logic. The preset behavior of the standard custom methods is to send/receive messages in XML format corresponding to the PAM business objects being posted or received, but you can overwrite this behavior with your own mapping logic.
    - Perform no mapping in the adapter at all. Instead, you can have the adapter operation accept and return variant data that is copied directly into or out of the message. The task of interpreting or parsing the message contents is, in this case, up to the private process designer, who might choose to do this in VBScript (NT) or JavaScript (NT and UNIX), in other extension actions, or in some other process logic. The unparsed operations perform this kind of data transformation.

- *Per-adapter-instance event, mapping, and network configuration* that lets you define properties in each adapter instance that contain information that the adapter needs for successful operation. This includes which Adapter Server event to return, which maps to apply, and how to connect to the MQSeries network (queue manager, channel, hostname, port, default queue). If necessary, you can override these values in the adapter implementation code so that each adapter instance can conform to a wide variety of MQSeries messaging infrastructures.
- *Low-level message acknowledgment* lets you automatically return low-level acknowledgment messages that might be of little or no functional interest to the process designer. You can customize the adapter to automatically return these acknowledgments, or to expect their arrival, without representing the acknowledgments as operation inputs or outputs. Custom methods allow you to implement this functionality.
- *Exposure of message headers and descriptive fields* that allows you to access the MQSeries Java Client MQMessage object, including the message descriptor fields. This allows further customization of messaging behavior beyond what is provided in the packaged classes.

- *QueueName and CorrelationID that are exposed to the private process,* which allows private process designers flexibility when communicating with MQSeries. It also avoids the necessity of writing custom Java code to specify message identification properties.
- *Operations that can all report informative exceptions and return status business objects*, which allows you to redirect exception text to a StatusBO instance. Each method in the prepackaged adapter code can throw a specialized MQSeriesAdapterException class, which includes useful information regarding the nature of errors.
- *Syncpoint control* treats all interactions with the queue manager, including sending and receiving messages, as a unit of work that can be committed or rolled back at any point.
- *Optional debug messaging,* which can be useful during deployment. As the adapter executes, debug-level messages can be directed to the Adapter Server shell.
- *MQSeries client tracing,* which allows you to set the level of MQSeries client tracing via an adapter property. The adapter writes traces to a file.
- *A template adapter type with sample custom code,* which illustrates the above features. You can use Adapter for MQSeries Messaging type, its Java implementation, and the associated custom Java code as a reference while you develop your own adapters.

TIP: Adapter for MQSeries Messaging includes Javadoc for all Java classes, which provides detailed descriptions of the behavior of these classes and their associated methods. You might want to refer to the Javadoc when you develop your own adapters.

About the Adapter for MQSeries Messaging environment

See *Installing Adapters for MQSeries* on page 5 for general system requirements. In addition to general requirements, Adapter for MQSeries Messaging requires the following software:

- IBM MQSeries classes for Java (MQ base Java), version 5.1.

Updating the CLASSPATH

Adapter for MQSeries Messaging requires that the IBM MQSeries classes for Java (MQ base Java) be installed on the computer where the Adapter Server is running. This software is separately available from IBM.

Note: The IBM SupportPac™, MA88: MQSeries classes for Java and MQSeries classes for Java Message Service, contains the required MQSeries software.

The following jar files must be installed (on both the PAM client and server computers):

- MQSeries\java\lib\com.ibm.mq.jar
- MQSeries\java\lib\com.ibm.mqbind.jar
- MQSeries\java\lib\com.ibm.mq.iiop.jar

For more information on IBM MQSeries Messaging products and software, see the IBM web site: http://www.software.ibm.com/ts/mqseries/support

After you install the IBM MQSeries classes for Java (MQ base Java), you must update your system CLASSPATH.

To update your CLASSPATH on Windows NT:

1. From the Windows Start menu, choose Settings > Control Panel > System.
2. Click the Environment tab.
3. Select the CLASSPATH system variable.

   The current CLASSPATH value appears.
4. Edit this field as necessary to include the class paths.

   For example, if the Java client components are installed in C:\mqm\java\lib, your CLASSPATH should include C:\mqm\java\lib\com.ibm.mq.jar, C:\mqm\java\lib\com.ibm.mqbind.jar, and C:\mqm\java\lib\com.ibm.mq.iiop.jar.
5. Click Set, and then click Apply.
6. Restart the Adapter Server so the new MQSeries Java classes can be recognized.

About configuring IBM MQSeries Messaging

Adapter for MQSeries Messaging is designed to require minimal setup in MQSeries.

- Adapter for MQSeries Messaging does not require its own local MQSeries resources (queues, channels, or queue managers).

  Because it uses the Java Client API, Adapter for MQSeries Messaging can connect to a queue manager on another computer. In fact, we recommend that the queue manager be remote; running both the queue manager and the Adapter Server on the same computer might adversely affect performance, especially in high-throughput environments. You can use an existing queue manager, or you can set up a dedicated queue manager if you choose, with alias and remote queues serving as the adapter's interface to the rest of the MQSeries network.

Tip: If you are using a non-TCP network, you need to set up a new queue manager to act as a proxy between the TCP and non-TCP networks.

- Adapter for MQSeries Messaging relies on any queue managers it accesses to be up and running, with all channels and queues properly defined.

  Attempts to communicate with a queue manager that is down, with an undefined or full queue, or with a disabled or non-existent channel, results in exceptions being reported. The adapter does not enable or actively troubleshoot an MQSeries network—beyond reporting these types of exceptions.
- When Adapter for MQSeries Messaging encounters an exception condition in MQSeries, the MQSeriesAdapterException contains the exact exception text returned by MQSeries.

  This exception text normally contains an error code number and short description. For more information on error conditions, see the *IBM MQSeries Messages* reference.

Installing Adapter for MQSeries Messaging

You must install Adapter for MQSeries Messaging—on the same computer where the Adapter Server is running—by placing the appropriate java and class files in your PAM partner directory.

Before you install Adapter for MQSeries Messaging, you must install and configure the MQSeries queue manager. Although you can run both the MQSeries queue manager and the Adapter Server on the same computer, we strongly recommend that you install them on different computers. The two computers must be able to communicate via TCP/IP.

Configuring the MQSeries queue manager requires defining a server connection channel, defining one or more local queues, and starting the channel listener. The queue manager might already exist in your enterprise, or you might need to create a new one for PAM.

Make sure that the IBM MQSeries classes for Java (MQ base Java) are installed on the computer where the Adapter Server is running, and that the Adapter Server's CLASSPATH is set to include the Java Client class files (see *Updating the CLASSPATH* on page 13).

IMPORTANT: Make sure you *back up* your partner root directory *before* installing the adapter. This is especially important if you have ever used (and customized) previous versions of MQSeries adapters.

To install Adapter for MQSeries Messaging:

1 Log on—as a user with administrative privileges—to the computer where the Adapter Server is installed.

2 Install Adapter for MQSeries Messaging.

On Windows NT:

A. Copy MQSeriesMessaging2_1.zip to your partner root directory.

B. Extract the zip file into your partner root directory.

IMPORTANT: If you previously installed an MQSeries adapter, a Confirm File Overwrite dialog appears. Click Yes to All (twice) to replace the existing files.

NOTE: If you have the PAM client and server on two different computers, you need to extract the files into the partner directory on both computers.

Adapter for MQSeries Messaging files appear in this location:

On this platform:	The files are located in:
Windows NT	<partner_root>\com\extricity\adapters\ibm\mqseries\test

Make sure that the java and class files are present in the same directory.

To import key elements:

For general importing guidelines, see *Importing key elements* on page 7.

▶ Move the file named MQSeriesMsgOptionsBO.xml from the Process Server computer to the computer where you'll be running the PAM client.

On this platform:	The files are located in:
Windows NT	<partner_root>\com\extricity\adapters\ibm\mqseries\test

Configuring Adapter for MQSeries Messaging

After you install Adapter for MQSeries Messaging, you can configure it and take advantage of its default behavior, or you can modify the adapter to suit your specific needs (see *Modifying Adapter for MQSeries Messaging* on page 18).

To configure Adapter for MQSeries Messaging:

1 Implement any custom code by writing one or more MQSeriesAdapterCustom subclasses.

Tip: If you plan to use the default mapping behavior of converting business objects to/from XML, do not edit the MQSeriesAdapterCustom class. See *Modifying Adapter for MQSeries Messaging* on page 18 for tips on implementing custom code.

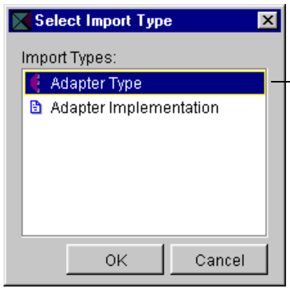
2 If the Adapter Server is not already running, start it.

3 Start the Partner Agreement Manager Process Server and the Adapter Server Administrator.

4 Choose Adapter Designer from the Tools menu to open the Adapter Designer.

5 In the Adapter Designer, choose Import from the File menu.

The Select Import Type dialog box appears.



You can import either an adapter type or an adapter implementation.

6 Select Adapter Type and click OK. In the Import dialog box, select the MQSeries Adapter Type XML file located in: <partner_root>\com\extricity\adapters\ibm\mqseries\test

This creates a template adapter type that you can edit to conform to your customized adapter code. For example, you can add any new operations you need.

- If you plan to use the mapped operations, you can edit their inputs and outputs to match the business objects you plan to get and post.
- Follow the instructions in the *Partner Agreement Manager Adapter Developer's Guide* to set default property values as appropriate for the adapter type.
- If you plan to use mapped events (events containing business object data), create a corresponding event type.

7 In the Adapter Designer, choose Import again, select Adapter Implementation, and select the MQSeries Adapter Java Imp XML file located in: <partner_root>\com\extricity\adapters\ibm\mqseries\test

This creates the adapter implementation. Make sure the adapter implementation's class name and package names are correct.

8 In the Adapter Server Administrator, choose Adapter Manager from the Tools menu to start the Adapter Manager.

9 Choose Add from the Adapter menu to create a new instance of Adapter for MQSeries Messaging, and follow the instructions in the *Partner Agreement Manager Adapter Developer's Guide* to set the connection property values.

- If you plan to use a command-line-executable map for your mapped operations, you must enter the system commands in the IncomingMap and OutgoingMap properties. Make sure to enter full pathnames.

- If your adapter checks for events, make sure that event polling is enabled and polls at an appropriate time interval.
- If you have created a subclass of the MQSeriesAdapterCustom class for this instance, you must enter its name in the CustomClass property.
- Set the DebugMessaging and MQSeriesTrace properties according to your debugging needs. You might want to turn these on only during development, testing, or maintenance. The logs associated with these properties can quickly grow very large.
- In the Adapter Manager, start the new adapter instance and resolve any exceptions that occur.

As soon as the adapter instance has connected to the queue manager, you are ready to begin using it in extension actions.

Modifying Adapter for MQSeries Messaging

Although you can use Adapter for MQSeries Messaging as is, the default behavior and settings (the name of the default queue, for example) might not meet your business needs. In addition, the default adapter instance communicates with only one queue. Therefore, you might want to customize Adapter for MQSeries Messaging to suit your specific needs.

Important: See the *Partner Agreement Manager Adapter Developer's Guide* for more information about customizing adapters.

The following are tips to bear in mind during implementation:

- Customize the operations in the adapter type to make it easier for process designers to select operations correctly.

  The PAM process builders who will use Adapter for MQSeries Messaging in private process extension actions rely on the names of the operations and their inputs and outputs to identify the correct operation. The operations provided with the Adapter for MQSeries Messaging type are intended as templates for your own specialized operations. Therefore, it's a good idea to change the generic GetMapped operation name to something more meaningful, such as GetPurchaseOrder. It's also a good idea to remove operations that you don't expect process designers to use. For example, if you do not expect to use unparsed messages, you might want to remove them from your adapter type.

- Make sure you understand your MQSeries network topology, its performance constraints, and its security rules so that you can set up specialized queue managers for PAM as appropriate.

  Normally, applications in an MQSeries network might have dedicated queue managers, or need to use queue managers set up as network proxies. For example, if PAM needs to communicate with an SNA network, you need a proxy queue manager to act as a bridge between SNA and TCP (the only protocol that Adapter for MQSeries Messaging can use). Also, if you expect PAM to send and receive a large number of messages, setting up a dedicated queue manager might be a more scalable alternative than simply reusing an existing queue manager, which might already experience heavy throughput.

- Make sure you understand the trade-offs between using mapped and unparsed operations.

  Mapped operations allow you to customize the transformation of data between PAM business objects and MQSeries messages. This allows PAM process designers to use the exact business objects that they want when defining private process extension actions. However, building these custom transformations involves development and ongoing maintenance efforts. On the other hand, using unparsed operations pushes business object transformation responsibility, as well as its maintenance, to the private process, where it may normally be implemented using VBScript (NT) or JavaScript (NT and UNIX), a different extension action, or other process-level logic.

- Make sure you implement the appropriate Adapter Server events.

  For example, the checkForEvents method included with Adapter for MQSeries Messaging merely checks for the availability of a message on a given queue. Although this is sufficient in most cases, you might want to implement a Request-Reply model, in which the adapter sends a message to some other application and awaits a response. If so, you need to implement this separately, possibly by reusing the Request operation logic.

- Review template operations provided in Adapter for MQSeries Messaging and add new operations as necessary.

The template operations provided with Adapter for MQSeries Messaging represent only a portion of the full range of operations you can implement using the Core class public methods. For example, although the template operations call the Core class getMessage and/or sendMessage functions only once (if at all), you can include operations that call these functions more than one time within an operation. This allows you to encapsulate more complex messaging interactions within a single operation, such as a single PAM business object to multiple MQSeries messages.

For example, a PAM process might include a purchase order business object that corresponds to several MQSeries messages—one for the header, plus additional messages for each individual order line. The easiest way to implement this is to add a new basic operation—"Get Purchase Order"—that calls getMessage multiple times in unparsed mode and copies information from the resulting variants into your Purchase Order business object.

Note: If you must run a queue manager and the Adapter Server on the same computer, we recommend a minimum of a 400 MHz processor computer with 256 MB or more RAM. You can improve performance by setting the Hostname property to `localhost`.

For more information about using or implementing the MQSeriesAdapterCore and MQSeriesAdapterCustom classes, see the accompanying Javadoc reference. You might also want to review the MQSeriesCustomExample subclass, which illustrates how to implement the various custom methods contained in the MQSeriesAdapterCustom class.

Testing Adapter for MQSeries Messaging

As a minimum, an adapter instance must meet these criteria before you can use it in an extension action. You must be able to:

- Start up and shut down the adapter instance without error.
- Verify that you can connect to the queue manager.

Before you release an adapter for general use by process designers, it's a good idea to test any custom code you have implemented. The easiest way to test custom code is to create a sample Partner Agreement Manager private process that uses extension actions that execute the adapter operations.

To test Adapter for MQSeries Messaging:

1. Create a PAM private process that includes attempts to post and get a series of business objects. Make sure that you execute every path in your custom code.
2. After getting a business object, make sure—in VBScript (NT) or JavaScript (NT and UNIX)—that the contents are as you expect.

   TIP: Use the StatusBO operation output and check the results.

3. Review the log file to make sure that no exceptions were reported within the adapter itself.
4. While debugging, try setting the adapter's DebugMessaging property to Terse or Verbose.
5. Turn on MQSeries tracing and look at the MQSeriesTrace.log file (located in the partner root directory) in the event of any MQSeries exceptions.
6. Test the public process events.

Adapter for MQSeries Messaging reference

This section describes in more detail the sample Adapter for MQSeries Messaging type that comes packaged with the product. The class names, properties, operations, and events are all described in the table below. Note that the operations are intended as templates only. You are encouraged to rename these operations and their parameters as needed, or design your own operations from scratch that use the MQSeriesAdapterCore class's methods in other ways.

NOTE: Each operation has a status BO that returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.

Adapter for MQSeries Messaging type

These are the components of the Adapter for MQSeries Messaging type.

Class name	com.extricity.adapters.ibm.mqseries.MQSeriesAdapter
Description	This class represents the adapter implementation. It is generated by the Adapter Designer and subsequently edited to call methods in the MQSeriesAdapterCore class.

Class name	com.extricity.adapters.ibm.mqseries.MQSeriesAdapterCore
Description	This represents the bulk of the MQSeries logic. It includes the following features: ■ Connect to and disconnect from queues and queue managers given connection information specified in the instance's properties or passed as parameters to given methods. ■ Dispatch the task of mapping between business objects and MQSeries messages. ■ Send and receive messages to/from queues. ■ Report exceptions containing meaningful error text in case error conditions are found. ■ Execute maps as specified in properties. ■ Call methods in the Custom class. ■ Handle transaction contexts via commit and rollback methods.
Class name	com.extricity.adapters.ibm.mqseries.MQSeriesAdapterException
Description	This exception subclass can be thrown by most methods, and contains information on error conditions. It also differentiates between those exceptions that are thrown by the MQSeries Java Client and those that are thrown by adapter code.
Class name	com.extricity.adapters.ibm.mqseries.MQSeriesRFH
Description	Base class for managing MQSeries Rules and Format Headers. This class might be helpful when interacting with MQSeries-enabled products such as MQSeries Integrator or MQSeries Publish/ Subscribe. Further refinements of this class are provided with Adapter for MQSeries Integrator and Adapter for MQSeries Publish/ Subscribe.
Class name	com.extricity.adapters.ibm.mqseries.MQSeriesAdapterCustom
Description	This class contains stub methods that can be implemented on site. These methods are as follows: ■ sendReply. Send a low-level acknowledgment to an incoming message. ■ receiveReply. Await a low-level acknowledgment to an outgoing message. ■ callInMap. Custom data transformation method for incoming messages; called whenever a "mapped" operation is indicated and the IncomingMap property is left null. ■ callOutMap. Custom data transformation method for outgoing messages; called whenever a "mapped" operation is indicated and the OutgoingMap property is left null. The default behavior of the mapping methods is to convert business objects to/from XML.

Properties

QueueManager	Name of queue manager to connect to. **Mandatory**.
Channel	Name of server connection channel by which to connect to queue manager. **Mandatory.**
DefaultQueue	Name of a default queue to use for sending and receiving messages. Can be overridden in code or in MQSeriesMsgOptions operation argument. **Optional.**
Hostname	Name of host on which QueueManager resides. This can be an IP address or the IP host name. **Mandatory**.
Port	Name of port on which channel is listening. Defaults to 1414. **Mandatory**.
UserID	The user name used to connect to the queue manager. **Optional**, but must be entered if security is enabled on the server connection channel.
Password	The password associated with the user name. **Optional**, but must be entered if security is enabled on the server connection channel.
CorrelationID	The value is used only during event checking (that is, checkForEvents); otherwise, it is ignored. **Optional**.
MapWorking Directory	The location in the local file system to use for writing and reading files during a command-line mapping execution. Ignored if IncomingMap and OutgoingMap are blank. **Optional**.

IncomingMap	Command line that must be executed to map incoming messages. **Optional.** If left blank, the adapter calls the callInMap custom method instead for all incoming mapped messages. If populated, then mapped messages undergo the following steps: 1. Write incoming MQSeries message contents to a file, <MapWorkingDirectory>\inMsg.txt (Windows NT) 2. Call the property value as a system command, which translates the MQSeries file to an XML file, <MapWorkingDirectory>\outBO.xml (Windows NT) 3. Read in the XML file and fill a business object with the contents.
OutgoingMap	Command line that must be executed to map outgoing messages. **Optional.** If left blank, the adapter calls the callOutMap custom method instead for all outgoing mapped messages. If populated, mapped messages undergo the following steps: 1. Write BO to an XML file, <MapWorkingDirectory>\inBO.xml 2. Call the property value as a system command, which translates the XML file to an MQSeries message file, <MapWorkingDirectory>\outMsg.txt 3. Read in the MQSeries file and put contents into an MQSeries message.
MQSeriesTraceLevel	Sets the MQSeries trace level. One of six values: Off, or one of the five levels of detail that the MQSeries trace facility records debug-level information to the file MQSeriesTrace.log, located in the partner root directory. **Mandatory**.
WaitInterval	The length of time, in milliseconds, that an attempt to get a message from a queue must wait before erroring out, if a message is not immediately available. **Mandatory**. Can be overridden in code.

	DebugMessaging	One of three values: Off, Terse, Verbose, representing different levels of debug messaging. Messages are written to the Adapter Server console. Debug messaging might be useful during development and deployment, but is generally set to Off in a production environment. **Mandatory**.
	CustomClass	Name of Custom class. Must be the fully qualified class pathname, dot-delimited, relative to the classpath. Defaults to com.extricity.adapters.ibm.mqseries.MQSeriesAdapterCustom. **Mandatory**.
Operations	**GetUnparsed**	Retrieves a message and returns raw contents to the private process. Message is consumed in the queue.
	Input	MQSeriesMsgOptions business object containing the value for the QueueName to be used by the operation.
	Output	Variant, representing message contents. Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
	BrowseUnparsed	Retrieves a message and returns raw contents to the private process. Message is not consumed in the queue.
	Input	MQSeriesMsgOptions business object containing the value for the QueueName to be used by the operation.
	Output	Variant, representing message contents. Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
	GetMapped	Retrieves a message, calls a map, and returns the business object. Consumes message in the queue. **NOTE:** If the IncomingMap property is specified, the operation writes the incoming message to a file, calls the command specified in this property, reads in the resulting XML, and returns the BO. If this property is left null, calls the Custom callInMap method.

Input	MQSeriesMsgOptions business object containing the value for the QueueName to be used by the operation.
Output	Business object. Also, status BO.
BrowseMapped	Retrieves a message, calls a map, and returns the business object. Does not consume message in the queue. **Note:** If the IncomingMap property is specified, the operation writes the incoming message to a file, calls the command specified in this property, reads in the resulting XML, and returns the BO. If this property is left null, calls the Custom callInMap method.
Input	MQSeriesMsgOptions business object containing the value for the QueueName to be used by the operation.
Output	Business object. Also, status BO.
PostUnparsed	Send a message, given the raw message contents.
Input	Variant, representing message contents. MQSeriesMsgOptions business object containing the value for the QueueName to be used by the operation.
Output	Status BO.
PostMapped	Send a message using map to do data transformation. **Note:** If the OutgoingMap property is specified, the operation writes the BO as an XML file, calls the command specified in this property, reads the resulting bytes into a new MQSeries message, and sends it. If this property is left null, calls the Custom callOutMap method.
Input	Business object. MQSeriesMsgOptions business object containing the value for the QueueName to be used by the operation.
Output	Status BO.
RequestUnparsed	Send a message (unparsed), then wait for a return message (unparsed).

	Input	Variant, representing contents of outgoing message. MQSeriesMsgOptions business object containing the value for the QueueName to be used by the operation. ReplyQueueName variant, an optional field, which specifies the queue from which the reply message is retrieved.
	Output	Variant, representing contents of reply message. Also, status BO.
	RequestMapped	Send a message (mapped), then wait for a return message (mapped).
	Input	Business object corresponding to output message. Map is applied to this BO as in PostMapped. MQSeriesMsgOptions business object containing the value for the QueueName to be used by the operation. ReplyQueueName variant, an optional field, which specifies the queue from which the reply message is retrieved.
	Output	Business object corresponding to reply message. Map is applied to this BO as in GetMapped. Also, status BO.
Events	MQEvent. Contains Variant data. Functionality of checkForEvents is to poll a given queue and return any messages that appear, as in GetUnparsed.	

NOTE: Each operation has a status BO that returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.

CHAPTER

4

Using Adapter for MQSeries Integrator (RFH)

Read this chapter to learn how to configure and use WebSphere® Partner Agreement Manager Adapter for MQSeries® Integrator (RFH) version 2.1.

This chapter includes the following sections:

Adapter for MQSeries Integrator (RFH) combines the message routing and transformation features of IBM MQSeries Integrator with the business-to-business integration capabilities of WebSphere Partner Agreement Manager.

Note: Partner Agreement Connect (PAC) is a version of Partner Agreement Manager (PAM) designed for partners who do not initiate processes. Because the same core documentation set is used for both PAM and PAC, the documentation uses the term “Partner Agreement Manager” or “PAM” to refer to both products.

About IBM MQSeries Integrator (RFH)

IBM MQSeries Integrator works with MQSeries messaging, extending its basic connectivity and transport capabilities to provide a powerful message broker solution driven by business rules. Messages are formed, routed, and transformed according to defined rules.

IBM MQSeries Integrator is a data transformation and routing engine for MQSeries. Also called MQSI, it is divided into three components, all of which store rules and formats in a relational database.

- The *Formatter* allows users to define data formats as well as the maps between them with a graphical, drag-and-drop interface.
- The *Rules Editor* allows users to specify, again in a graphical environment, which maps to apply to incoming messages depending on their formats and contents, as well as to which MQSeries queues to route the resulting outbound messages.
- The *Rules Engine* connects to an MQSeries queue manager and processes incoming messages according to the rules and formats defined in the Formatter and Rules Editor.

IBM MQSeries Integrator also includes command line utilities and scripts designed to facilitate development and on-site testing.

The Rules Engine recognizes incoming MQSeries messages as being of given formats by either:

- Scanning the full message and applying it to stored formats in a declarative manner.
- Reading the format identifiers in the message’s MQSeries Rules and Format Header (MQRFH).

The MQRFH is a well-defined segment at the beginning of the MQSeries message's contents section. It provides information to the Rules Engine about the message, including its format, string representation, message length, and so forth. The presence of an MQRFH header in a message often improves processing times.

Adapter for MQSeries Integrator (RFH) provides support to adapter developers who must work with messages that include MQRFH headers. It also includes template adapter operations and sample custom code that illustrates how to manipulate and further customize MQRFH headers by way of an intuitive API.

About Adapter for MQSeries Integrator (RFH)

This version of Adapter for MQSeries Integrator (RFH) is compatible with WebSphere Partner Agreement Manager (PAM) version 2.1. It requires the IBM MQSeries classes for Java (MQ base Java)™, which is compatible with all versions of MQSeries through version 5.1. Adapter for MQSeries Integrator (RFH) is compatible with IBM MQSeries Integrator version 1.0/ 1.1.

Adapter for MQSeries Integrator (RFH) contains the following:

- class files corresponding to all Java classes.
- Java files for MQSeriesIntegratorAdapter and MQSeriesIntegratorAdapterCustom classes.
- Java and class files for an example MQSeriesIntegratorAdapterCustom subclass (MQSeriesIntegratorCustomExample.java and MQSeriesIntegratorCustomExample.class).
- XML representations of template adapter type and implementation (MQSeriesIntegratorAdapterType.xml, MQSeriesIntegratorAdapterJavaImp.xml).
- Javadoc reference for all class files.

Key features of Adapter for MQSeries Integrator (RFH) include:

- *Adapter for MQSeries Integrator (RFH) implementations are subclassed from the base Adapter for MQSeries Messaging.* With the exception of managing the MQRFH, the sending and receiving of messages is the same in both adapters.
- *Customizable classes allow you to manage the MQRFH.*

- *Templates* provide for specialized MQSeriesIntegrated send, receive, and request-reply operations.
- *The adapter provides an ability to construct and send administrative messages to the Rules Engine*, including shutting down and reloading rule sets from the database.

Tip: Adapter for MQSeries Integrator (RFH) includes Javadoc for all Java classes, which provides detailed descriptions of the behavior of these classes and their associated methods. You might want to refer to the Javadoc when you develop your own adapters.

About the Adapter for MQSeries Integrator (RFH) environment

See *Installing Adapters for MQSeries* on page 5 for general system requirements. In addition to general requirements, Adapter for MQSeries Integrator (RFH) requires the following software:

- IBM MQSeries classes for Java (MQ base Java), version 5.1.

Updating the CLASSPATH

Adapter for MQSeries Integrator (RFH) requires that the IBM MQSeries classes for Java (MQ base Java) be installed on the computer where the Adapter Server is running. This software is separately available from IBM.

Note: The IBM SupportPac™, MA88: MQSeries classes for Java and MQSeries classes for Java Message Service, contains the required MQSeries software.

The following jar files must be installed (on both the PAM client and server computers):

- `MQSeries\java\lib\com.ibm.mq.jar`
- `MQSeries\java\lib\com.ibm.mqbind.jar`
- `MQSeries\java\lib\com.ibm.mq.iiop.jar`

For more information on IBM MQSeries Integrator products and software, see the IBM web site:

http://www.software.ibm.com/ts/mqseries/support

After you install the IBM MQSeries classes for Java (MQ base Java), you must update your system CLASSPATH.

To update your CLASSPATH on Windows NT:

1 From the Windows Start menu, choose Settings > Control Panel > System.

2 Click the Environment tab.

3 Select the CLASSPATH system variable.

   The current CLASSPATH value appears.

4 Edit this field as necessary to include the class paths.

   For example, if the Java client components are installed in `C:\mqm\java\lib`, your CLASSPATH should include `C:\mqm\java\lib\com.ibm.mq.jar`, `C:\mqm\java\lib\com.ibm.mqbind.jar`, and `C:\mqm\java\lib\com.ibm.mq.iiop.jar`.

5 Click Set, and then click Apply.

6 Restart the Adapter Server so the new MQSeries Java classes can be recognized.

About configuring IBM MQSeries Integrator (RFH)

Adapter for MQSeries Integrator (RFH) is designed to require minimal setup in MQSeries.

- An MQSeries adapter relies on any queue managers it accesses to be up and running, with all channels and queues properly defined. The queue manager must be accessible via the IBM MQSeries classes for Java (MQ base Java), it must have an MQSeries Integrator Rules Engine running against it, and the MQSI-specific queues (input, outputs, no-hit, failure) must be defined correctly.

Important: Be sure to edit the MQSeriesIntegratorAdapter.java file to enter the input and output queue names in the appropriate queueName variables.

The adapter reports an exception if you attempt to communicate with a queue manager that is down, with an undefined or full queue, or with a disabled or non-existent channel. The adapter doesn't enable or actively troubleshoot an MQSeries network—beyond reporting these types of exceptions.

If the Rules Engine is inoperative, sending messages to it results in those messages queueing up without any feedback. Attempts to receive messages fail in this case. In general, the adapter does not actively troubleshoot an inoperative or otherwise improperly functioning MQSeries queue manager or MQSeries Integrator Rules Engine, beyond reporting on exceptions or time-outs encountered.

- When the adapter encounters an exception condition in MQSeries, the MQSeriesAdapterException contains the exact exception text returned by MQSeries.

  This exception text contains an error code number and short description. For more information on possible error conditions, see chapter 5 of the *IBM MQSeries Application Programming Reference*.

Installing Adapter for MQSeries Integrator (RFH)

You must install Adapter for MQSeries Integrator (RFH)—on the same computer where the Adapter Server is running—by copying the appropriate java and class files into your PAM partner directory.

Before you install Adapter for MQSeries Integrator (RFH), you must install and configure the MQSeries queue manager. Although you can run both the MQSeries queue manager and the Adapter Server on the same computer, we strongly recommend that you install them on different computers. The two computers must be able to communicate via TCP/IP.

Configuring the MQSeries queue manager includes defining a server connection channel, defining one or more local queues, and starting the channel listener. The queue manager might already exist in your enterprise, or you might need to create a new one for Partner Agreement Manager.

Make sure that:

- the MQSeries Integrator Rules Engine is running against the correct queue manager, that rules and formats are properly defined, and that MQSI-specific queues are set up. By default, the adapter sends messages intended for MQSeries Integrator to a queue named MQSI.INPUT, and receives messages from IBM MQSeries Integrator in a queue named MQSI.OUTPUT.
- the IBM MQSeries classes for Java (MQ base Java) are installed on the computer where the Adapter Server is running, and that the Adapter Server's CLASSPATH is set to include the Java client class files (see *Updating the CLASSPATH* on page 32).

IMPORTANT: Make sure you *back up* your partner root directory *before* installing the adapter. This is especially important if you have ever used (and customized) previous versions of MQSeries adapters.

To install Adapter for MQSeries Integrator (RFH):

1. Log on—as a user with administrative privileges—to the computer where the Adapter Server is installed.

   Make sure the Process Server and Adapter Server are not running.

2. Install Adapter for MQSeries Integrator (RFH).

   On Windows NT:

   A. Copy MQSeriesIntegrator2_1.zip to your partner root directory.

   B. Extract the zip file into your partner root directory.

IMPORTANT: If you previously installed an MQSeries adapter, a Confirm File Overwrite dialog appears. Click Yes to All (twice) to replace the existing files.

NOTE: If you have the PAM client and server on two different computers, you need to extract the files into the partner directory on both computers.

Adapter for MQSeries Integrator (RFH) files appear in this location:

On this platform:	The files are located in:
Windows NT	<partner_root>\com\extricity\adapters\ibm\mqseries\mqsi2

Make sure that the java and class files are present in the same directory.

To import key elements:

For general importing guidelines, see *Importing key elements* on page 7.

- Move the file named MQSeriesMsgOptionsBO.xml from the Process Server computer to the computer where you're running the PAM client.

On this platform:	The files are located in:
Windows NT	<partner_root>\com\extricity\adapters\ibm\mqseries\mqsi2 \test

Configuring Adapter for MQSeries Integrator (RFH)

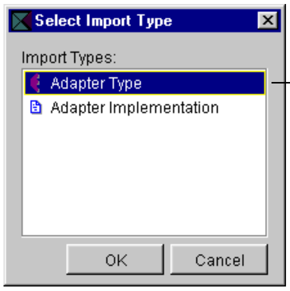
After you install Adapter for MQSeries Integrator (RFH), you can configure it and take advantage of its default behavior, or you can modify the adapter to suit your specific needs (see *Modifying Adapter for MQSeries Integrator (RFH)* on page 38).

To configure Adapter for MQSeries Integrator (RFH):

1. Implement any custom code by writing one or more MQSeriesAdapterCustom classes or subclasses.

   See *Modifying Adapter for MQSeries Integrator (RFH)* on page 38 for tips on implementing custom code.
2. If the Adapter Server is not already running, start it.
3. Start the Partner Agreement Manager Process Server and the Adapter Server Administrator.
4. Choose Adapter Designer from the Tools menu to open the Adapter Designer.
5. In the Adapter Designer, choose Import from the File menu.

The Select Import Type dialog box appears.



You can import either an adapter type or an adapter implementation.

6 Select Adapter Type and click OK. In the Import dialog box, select the MQSeriesIntegratorAdapterType XML file located in:
`<partner_root>\com\extricity\adapters\ibm\mqseries\mqsi2\test`

This creates a template adapter type that you can edit to conform to your customized adapter code. For example, you can add any new operations you need.

- If you plan to use the MQSI or mapped operations, you can edit their inputs and outputs to match the business objects you plan to get and post.
- Follow the instructions in the *Partner Agreement Manager Adapter Developer's Guide* to set default property values as appropriate for the adapter type.
- If you plan to use mapped events (events containing business object data), create a corresponding event type.

7 In the Adapter Designer, choose Import again, select Adapter Implementation, and select the MQSeriesIntegratorAdapterJavaImp XML file located in:
`<partner_root>\com\extricity\adapters\ibm\mqseries\mqsi2\test`

This creates the adapter implementation. Make sure the adapter implementation's class name and package names are correct.

8 In the Adapter Server Administrator, choose Adapter Manager from the Tools menu to start the Adapter Manager.

9 Choose Add from the Adapter menu to create a new instance of Adapter for MQSeries Integrator (RFH), and follow the instructions in the *Partner Agreement Manager Adapter Developer's Guide* to set the connection property values.

- If you plan to use a command-line-executable map for your mapped operations, you must enter the system commands in the IncomingMap and OutgoingMap properties.
- If your adapter checks for events, make sure that event polling is enabled and polls at an appropriate time interval.
- If you have created a subclass of the MQSeriesIntegratorAdapterCustom class for this instance, you must enter its name in the CustomClass property.
- Set the DebugMessaging and MQSeriesTrace properties according to your debugging needs. You might want to turn these on only during development, testing, or maintenance. The logs associated with these properties can quickly grow very large.
- In the Adapter Manager, start the new adapter instance and resolve any exceptions that occur.

As soon as the adapter instance has connected to the queue manager, you are ready to begin using it in extension actions.

Modifying Adapter for MQSeries Integrator (RFH)

Although you can use Adapter for MQSeries Integrator (RFH) as is, the default behavior and settings might not meet your business needs. For example, the default mapping behavior for mapped operations is to convert business objects to/from XML with MQRFH. In addition, MQSI operations are preset to send messages to a queue named MQSI.INPUT and receive messages from a queue named MQSI.OUTPUT. Other operations are preset to connect to the queue that is specified in the DefaultQueue property.

Also, remote applications receiving your MQSeries messages need to understand XML and be able to construct the XML messages that your adapter expects.

As a result, you might want to customize Adapter for MQSeries Integrator (RFH) to suit your specific needs.

Important: See the *Partner Agreement Manager Adapter Developer's Guide* for more information about customizing adapters.

The following are tips to bear in mind during implementation:

- Customize the operations in the adapter type to make it easier for process designers to select operations correctly.

  The PAM process builders who will use this adapter in private process extension actions rely on the names of the operations and their inputs and outputs to identify the correct operation. The operations provided with the Adapter for MQSeries Integrator (RFH) type are intended as templates for your own specialized operations. Therefore, it's a good idea to change the generic PostMQSI operation name to something more meaningful, such as PostPurchaseOrder. It's also a good idea to remove operations that you don't expect process designers to use. For example, if you do not expect to use unparsed messages, you might want to remove them from your adapter type.

- Review template operations provided in Adapter for MQSeries Integrator (RFH) and add new operations as necessary.

  The template operations provided with Adapter for MQSeries Integrator (RFH) represent only a portion of the full range of operations you can implement using the Core class public methods combined with MQRFH class methods. For example, although the template operations call the Core class getMessage and/or sendMessage functions only once (if at all) and create generic MQRFH instances, you can include operations that call these functions more than one time within an operation. This allows you to encapsulate more complex messaging interactions within a single operation, such as a single PAM business object to multiple MQSeries messages.

  For example, a PAM process might include a purchase order business object that corresponds to several MQSeries messages—one for the header, plus additional messages for each individual order line. The easiest way to implement this would be to add a new basic operation—"Get Purchase Order"—that could call getMessage multiple times in unparsed mode and copy information from the resulting variants into your Purchase Order business object.

TIP: Before you begin writing custom Java code for your adapter that deals with messages to/from MQSI, you need to have a solid understanding of the formats defined in MQSI, which applications require those formats, and how the Rules Engine is mapping messages and routing them to their destinations. Information you need to have before coding includes the Application Group and Message Type of your messages, their formats, and the queues on which they appear.

For more information about using or implementing the MQSeriesAdapterCore, MQSeriesIntegratorRFH, and MQSeriesIntegratorAdapterCustom classes, see the accompanying Javadoc reference. You might also want to review the MQSeriesIntegratorCustomExample subclass, which illustrates how to implement the various custom methods contained in the MQSeriesAdapterCustom class.

Testing Adapter for MQSeries Integrator (RFH)

As a minimum, an adapter instance must meet these criteria before you can use it in an extension action. You must be able to:

- start up and shut down the adapter instance without error.
- verify that you can connect to the queue manager.

Before you release an adapter for general use by process designers, it's a good idea to test any custom code you have implemented. The easiest way to test custom code is to create a sample Partner Agreement Manager private process that uses extension actions that execute the adapter operations.

To test Adapter for MQSeries Integrator (RFH):

1. Create a PAM private process that includes attempts to post and get a series of business objects. Make sure that you execute every path in your custom code.
2. After getting a business object, make sure—in VBScript (NT) or JavaScript (NT and UNIX)—that the contents are as you expect.

   Tip: Use the StatusBO operation output and check the results.

3. Review the log file to make sure that no exceptions were reported within the adapter itself.
4. While debugging, try setting the adapter's DebugMessaging property to Terse or Verbose.
5. Turn on MQSeries tracing and look at the MQSeriesTrace.log file (located in the partner root directory) in the event of any MQSeries exceptions.
6. Test the public process events.

Adapter for MQSeries Integrator (RFH) reference

This section describes in more detail the sample Adapter for MQSeries Integrator (RFH) type that comes packaged with the product. The class names, properties, operations, and events are all described in the table below.

Note: The operations are intended as templates only. You are encouraged to rename these operations and their parameters as needed, or design your own operations from scratch that use the MQSeriesAdapterCore class's methods in other ways.

Finally, since this sample adapter is a subclass of the PAM Adapter for MQSeries Messaging, the messaging adapter's operations are also included.

Note: Each operation has a status BO that returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.

Adapter for MQSeries Integrator (RFH) type

These are the components of Adapter for MQSeries Integrator (RFH) type.

Class name	com.extricity.adapters.ibm.MQSeries.mqsi.MQSeriesIntegrator Adapter
Description	This class represents the adapter implementation. It was generated from the Adapter Designer and subsequently edited to call methods in the MQSeriesAdapterCore class.
Class name	com.extricity.adapters.ibm.MQSeries.mqsi.MQSeriesIntegratorRFH
Description	Class for managing MQSeries Rules and Format Headers of messages sent to or received from the MQSI Rules Engine. This class hides the complexity of managing the MQRFH fields and commands that are specific to MQSI.
Class name	com.extricity.adapters.ibm.MQSeries.mqsi.MQSeriesIntegrator CustomExample

Description This class contains stub methods that can be implemented on site. These methods are as follows:

- sendReply. Send a low-level acknowledgment to an incoming message.
- receiveReply. Await a low-level acknowledgment to an outgoing message.
- callInMap. Custom data transformation method for incoming messages; called whenever a "mapped" operation is indicated and the IncomingMap property is left null.
- callOutMap. Custom data transformation method for outgoing messages; called whenever a "mapped" operation is indicated and the OutgoingMap property is left null.

The default behavior of the mapping methods is to convert business objects to/from XML as well as read/write default MQRFHs.

Properties

QueueManager	Name of queue manager to connect to. **Mandatory**.
Channel	Name of server connection channel by which to connect to queue manager. **Mandatory**.
DefaultQueue	Name of a default queue to use for sending and receiving messages. Can be overridden in code. **Optional**.
Hostname	Name of host on which QueueManager resides. This can be an IP address or the IP host name. **Mandatory**.
Port	Name of port to which Channel is listening. Defaults to 1414. **Mandatory**.
UserID	The user name used to connect to the queue manager. **Optional**, but must be entered if security is enabled on the server connection channel.
Password	The password associated with the user name. **Optional**, but must be entered if security is enabled on the server connection channel.
CorrelationID	The value is used only during event checking (that is, checkForEvents); otherwise, it is ignored. **Optional**.
MapWorking Directory	The location in the local file system to use for writing and reading files during a command-line mapping execution. Ignored if IncomingMap and OutgoingMap are blank. **Optional**.

IncomingMap	Command line that must be executed in order to map incoming messages. **Optional**. If left blank, the adapter calls the callInMap custom method instead for all incoming mapped messages. If populated, then mapped messages undergo the following steps: 1. Write incoming MQSeries message contents to a file, <MapWorkingDirectory>\inMsg.txt (Windows NT) 2. Call the property value as a system command, which translates the MQSeries file to an XML file, <MapWorkingDirectory>\outBO.xml (Windows NT) 3. Read in the XML file and fill a business object with the contents. **Note:** If this mapping feature is used for MQSI adapters, the map executable must be MQRFH-aware.
OutgoingMap	Command line that must be executed in order to map outgoing messages. **Optional.** If left blank, the adapter calls the callOutMap custom method instead for all outgoing mapped messages. If populated, mapped messages undergo the following steps: 1. Write BO to an XML file, <MapWorkingDirectory>\inBO.xml (Windows NT) 2. Call the property value as a system command, which translates the XML file to an MQSeries message file, <MapWorkingDirectory>\outMsg.txt (Windows NT) 3. Read in the MQSeries file and put contents into an MQSeries message. **Note:** If this mapping feature is used for MQSI adapters, the map executable must be MQRFH-aware.
MQSeriesTraceLevel	Sets the MQSeries trace level. One of six values: Off, or one of the five levels of detail that the MQSeries trace facility records debug-level information to the file MQSeriesTrace.log, located in the partner root directory. **Mandatory**.

	WaitInterval	The length of time, in milliseconds, that an attempt to Get a message from a queue must wait before an error is generated, if a message is not immediately available. –1 indicates waiting indefinitely. **Mandatory**. Can be overridden in code.
	DebugMessaging	One of three values: Off, Terse, Verbose, representing different levels of debug messaging. Messages are written to the Adapter Server console. Debug messaging can be useful during development and deployment, but is generally set to Off in a production environment. **Mandatory**.
	ApplicationGroup	The default MQSI Application Group to embed in the MQRFH of all outbound messages. Can be overridden in code. **Optional**.
	DefaultMessageType	The default MQSI Message Type to embed in the MQRFH of all outbound messages. Can be overridden in code. **Optional**.
	CustomClass	Name of example Custom class. Must be the fully qualified class pathname, dot-delimited, relative to the classpath. Defaults to com.extricity.adapters.ibm.mqseries. MQSeriesIntegratorCustomExample. **Mandatory**.
Operations	**GetUnparsed**	Retrieves a message and returns raw contents to the private process. Message is consumed in the queue.
	Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
	Output	Variant, representing message contents. Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
	GetMQSI	Retrieves a message, calls a map, including MQRFH, and returns the functional BO. NOTE: If the IncomingMap property is specified, the operation writes the incoming message to a file, calls the command specified in this property, reads in the resulting XML, and returns the BO. If this property is left null, then calls the custom map method.

Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	The BO specified in the adapter's type definition. Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
BrowseMQSI	Retrieves a message without consuming it in queue, calls a map, including MQRFH, and returns the functional BO. NOTE: If the IncomingMap property is specified, the operation writes the incoming message to a file, calls the command specified in this property, reads in the resulting XML, and returns the BO. If this property is left null, then calls the custom map method.
Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	The BO specified in the adapter type definition. Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
PostMQSI	Send a message using map to do data transformation, with MQRFH. NOTE: If the OutgoingMap property is specified, the operation writes the BO as an XML file, calls the command specified in this property, reads the resulting bytes into a new MQSeries message, and sends it. If this property is left null, then calls the custom map method.
Input	The BO specified in the adapter type definition. MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
RequestMQSI	Send a message (mapped), with MQRFH, then wait for a return message (mapped) with MQRFH

Input	The BO specified in adapter type definition. Map is applied to this BO as in PostMapped. A variant, which must be specified by the private process, indicating the queue name where the reply must be sent. MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	The BO specified in adapter type definition. Map is applied to this BO as in GetMapped. Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
BrowseUnparsed	Retrieves a message and returns raw contents to the private process. Message is not consumed in the queue.
Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Variant, representing message contents. Also, status BO.
GetMapped	Retrieves a message, calls a map, and returns the business object. Consumes message in the queue. **Note:** If the IncomingMap property is specified, the operation writes the incoming message to a file, calls the command specified in this property, reads in the resulting XML, and returns the BO. If this property is left null, calls the Custom callInMap method.
Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Business object. Also, status BO.
BrowseMapped	Retrieves a message, calls a map, and returns the business object. Does not consume message in the queue. **Note:** If the IncomingMap property is specified, the operation writes the incoming message to a file, calls the command specified in this property, reads in the resulting XML, and returns the BO. If this property is left null, calls the Custom callInMap method.

Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Business object. Also, status BO.
PostUnparsed	Send a message, given the raw message contents.
Input	Variant, representing message contents. MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Status BO.
PostMapped	Send a message using map to do data transformation. **Note:** If the OutgoingMap property is specified, the operation writes the BO as an XML file, calls the command specified in this property, reads the resulting bytes into a new MQMessage, and sends it. If this property is left null, calls the Custom callOutMap method.
Input	Business object. MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Status BO.
RequestUnparsed	Send a message (unparsed), then wait for a return message (unparsed).
Input	Variant, representing contents of outgoing message. A variant, which must be specified by the private process, indicating the queue name where the reply must be sent. MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Variant, representing contents of reply message. Also, status BO.
RequestMapped	Send a message (mapped), then wait for a return message (mapped).

	Input	Business object corresponding to output message. Map is applied to this BO as in PostMapped.
		A variant, which must be specified by the private process, indicating the queue name where the reply must be sent.
		MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
	Output	Business object corresponding to reply message. Map is applied to this BO as in GetMapped. Also, status BO.
Events	MQEvent. Contains Variant data. Functionality of checkForEvents is to poll a given queue and return any messages that appear, as in GetUnparsed.	

NOTE: Each operation has an optional status BO that returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.

CHAPTER

5

Using Adapter for MQSeries Integrator (RFH2)

Read this chapter to learn how to configure and use WebSphere® Partner Agreement Manager Adapter for MQSeries® Integrator (RFH2) version 2.1.

This chapter includes the following sections:

Adapter for MQSeries Integrator (RFH2) combines the message routing and transformation features of IBM MQSeries Integrator with the business-to-business integration capabilities of WebSphere Partner Agreement Manager.

NOTE: Partner Agreement Connect (PAC) is a version of Partner Agreement Manager (PAM) designed for partners who do not initiate processes. Because the same core documentation set is used for both PAM and PAC, the documentation uses the term "Partner Agreement Manager" or "PAM" to refer to both products.

About IBM MQSeries Integrator (RFH2)

IBM MQSeries Integrator works with MQSeries messaging, extending its basic connectivity and transport capabilities to provide a powerful message broker solution driven by business rules. Messages are formed, routed, and transformed according to defined rules.

IBM MQSeries Integrator is a data transformation and routing engine for MQSeries. Also called MQSI, it is composed of four major components, all of which store message and configuration data in a relational database.

- A *message broker* hosts and controls business processes, or message flows. Applications communicate with the broker for the services provided by the message flows. There can be any number of brokers within a broker domain.
- The *Configuration Manager* manages the components and resources—called a broker domain—in the configuration repository. It initializes brokers and message processing operations and checks the authority of defined user IDs.
- The *Control Center* provides a graphical user interface for creating, manipulating, and deploying a broker domain. It also allows for monitoring and managing the operational state of the broker domain.
- The *User Name Server* interfaces with the operating system to determine valid users and groups.

A message broker recognizes incoming MQSeries messages as being of given formats by either:

- Scanning the full message and applying it to stored formats in a declarative manner.
- Reading the format identifiers in the message's MQ rules and formatting header (MQRFH2).

The MQRFH2 is a well-defined segment at the beginning of the MQ message's contents section. It provides information to the message broker about the message, including its format, string representation, message length, and so forth. The presence of an MQRFH2 header in a message often improves processing times.

Note: Although IBM MQSeries Integrator also supports the MQMD header, it is not exposed to Adapter for MQSeries Integrator (RFH2) developers directly. For example, if you need to set MQMD properties, you can use an MQMessage object.

Adapter for MQSeries Integrator (RFH2) provides support to adapter developers who must work with messages that include MQRFH2 headers. It also includes template adapter operations and sample custom code that illustrates how to manipulate and further customize MQRFH2 headers by way of an intuitive API.

About Adapter for MQSeries Integrator (RFH2)

This version of Adapter for MQSeries Integrator (RFH2) is compatible with WebSphere Partner Agreement Manager (PAM) version 2.1. It requires the IBM MQSeries classes for Java (MQ base Java)™, which is compatible with all versions of MQSeries through version 5.1. Adapter for MQSeries Integrator (RFH2) is compatible with IBM MQSeries Integrator version 2.0.

Adapter for MQSeries Integrator (RFH2) contains the following:

- Java files for MQSeriesAdapter, MQSeriesAdapterCustom, MQSeriesCustomExample, and MQSeriesIntegratorAdapter, MQSeriesIntegratorCustomExample.
- class files corresponding to all Java classes.
- class files designed to handle the IBM MQRFH2 rules and formatting header, MQSeriesIntegratorRFH2.
- various other class and properties files, inherited from the PAM Adapter for MQSeries Messaging.
- Java and class files, MQSeriesIntegratorCustomExample, providing an example adapter, which you can use as a base for your own adapter customizations.

- XML representations of template adapter type, implementation, and instance (MQSeriesIntegrator V2Adapter Type.xml, MQSeriesIntegrator Adapter V2Java Imp.xml, MQSeriesIntegratorV2Adapter.xml).
- XML formatted business objects (BOs) including MQSeriesMsgOptionsBO.xml, MQSIv2HeaderBO.xml, and MQSIv2MessageBO.xml.
- Javadoc reference for class files.

Key features of Adapter for MQSeries Integrator (RFH2) include:

- *Adapter for MQSeries Integrator (RFH2) implementations are subclassed from the base Adapter for MQSeries Messaging,* which allows you to reuse your Adapter for MQSeries Messaging. With the exception of managing the MQRFH2, the sending and receiving of messages is the same in both adapters.
- *With customizable classes,* you can manage the MQRFH2 rules and formatting header.
- *Templates* for a specialized Adapter for MQSeries Integrator (RFH2) provide for send, receive, and request-reply operations.
- The adapter provides an *ability to check for events*, or MQSeries messages, which then initiate PAM public processes.

TIP: Adapter for MQSeries Integrator (RFH2) includes Javadoc for required Java classes, which provides detailed descriptions of the behavior of these classes and their associated methods. You might want to refer to the Javadoc when you develop your own adapters.

About the Adapter for MQSeries Integrator (RFH2) environment

See *Installing Adapters for MQSeries* on page 5 for general system requirements. In addition to general requirements, Adapter for MQSeries Integrator (RFH2) requires the following software:

- IBM MQSeries classes for Java (MQ base Java), version 5.1.

Updating the CLASSPATH

Adapter for MQSeries Integrator (RFH2) requires that the IBM MQSeries classes for Java (MQ base Java) be installed on the computer where the Adapter Server is running. This software is separately available from IBM.

Note: The IBM SupportPac™, MA88: MQSeries classes for Java and MQSeries classes for Java Message Service, contains the required MQSeries software.

The following jar files must be installed (on both the PAM client and server computers)

- MQSeries\java\lib\com.ibm.mq.jar
- MQSeries\java\lib\com.ibm.mqbind.jar
- MQSeries\java\lib\com.ibm.mq.iiop.jar

For more information on IBM MQSeries Integrator products and software, see the IBM web site:

http://www.software.ibm.com/ts/mqseries/support

After you install the IBM MQSeries classes for Java (MQ base Java), you must update your system CLASSPATH.

To update your CLASSPATH on Windows NT:

1. From the Windows Start menu, choose Settings > Control Panel > System.
2. Click the Environment tab.
3. Select the CLASSPATH system variable.

   The current CLASSPATH value appears.
4. Edit this field as necessary to include the class paths.

   For example, if the Java client components are installed in C:\mqm\java\lib, your CLASSPATH should include C:\mqm\java\lib\com.ibm.mq.jar, C:\mqm\java\lib\com.ibm.mqbind.jar, and C:\mqm\java\lib\com.ibm.mq.iiop.jar.
5. Click Set, and then click Apply.
6. Restart the Adapter Server so the new MQSeries Java classes will be recognized.

About configuring IBM MQSeries Integrator (RFH2)

Adapter for MQSeries Integrator (RFH2) is designed to require minimal setup in MQSeries. However, be aware of the following basic interactions between Adapter for MQSeries Integrator (RFH2) and MQSeries:

- Adapter for MQSeries Integrator (RFH2) relies on any queue managers it accesses to be up and running, with all channels and queues properly defined. The queue manager must be accessible via the MQSeries Java client, it must have an IBM MQSeries Integrator message broker running against it, and the MQSI-specific queues (input, outputs, no-hit, failure) must be defined correctly.
- The adapter reports an exception if you attempt to communicate with a queue manager that is down, with an undefined or full queue, or with a disabled or non-existent channel.
- If the message broker is inoperative, sending messages to it results in those messages queueing up without any feedback. Attempts to receive messages would fail in this case.
- In general, the adapter does not actively troubleshoot an inoperative or otherwise improperly functioning MQSeries queue manager or IBM MQSeries Integrator message broker, beyond reporting on exceptions or time-outs encountered.
- When the adapter encounters an exception condition in MQSeries, the MQSeriesAdapterException contains the exact exception text returned by MQSeries.

  This exception text normally contains an error code number and short description. For more information on possible error conditions, see the *IBM MQSeries Messages* reference.

Installing Adapter for MQSeries Integrator (RFH2)

You must install Adapter for MQSeries Integrator (RFH2)—on the same computer where the Adapter Server is running—by placing the appropriate java and class files into your PAM partner directory.

Before you install Adapter for MQSeries Integrator (RFH2), you must install and configure the MQSeries queue manager. Although you can run both the MQSeries queue manager and the Adapter Server on the same computer, we strongly recommend that you install them on different computers. The two computers must be able to communicate via TCP/IP.

Configuring the MQSeries queue manager includes defining a server connection channel, defining one or more local queues, and starting the channel listener. The queue manager might already exist in your enterprise, or you can create a new one.

Make sure that:

- The IBM MQSeries Integrator message broker is running against the correct queue manager, that message flows are properly defined, and that MQSI-specific queues are set up.
- The IBM MQSeries classes for Java (MQ base Java) are installed on the computer where the Adapter Server is running, and that the Adapter Server's CLASSPATH is set to include the Java Client class files (see *Updating the CLASSPATH* on page 53).

IMPORTANT: Make sure you *back up* your partner root directory *before* installing the adapter. This is especially important if you have ever used (and customized) previous versions of MQSeries adapters.

To install Adapter for MQSeries Integrator (RFH2):

1. Log on—as a user with administrative privileges—to the computer where the Adapter Server is installed.
2. Install Adapter for MQSeries Integrator (RFH2).

   On Windows NT:

   A. Copy MQSeriesIntegrator2_1.zip to your partner root directory.

   B. Extract the zip file into your partner root directory.

IMPORTANT: If you previously installed an MQSeries adapter, a Confirm File Overwrite dialog appears. Click Yes to All (twice) to replace the existing files.

Note: If you have the PAM client and server on two different computers, you need to extract the files into the partner directory on both computers.

Adapter for MQSeries Integrator (RFH2) files appear in this location:

On this platform:	The files are located in:
Windows NT	<partner_root>\com\extricity\adapters\ibm\mqseries\mqsi2

Make sure that the java and class files are present in the same directory.

To import key elements:

For general importing guidelines, see *Importing key elements* on page 7.

1. Locate the business objects: MQSeriesMsgOptionsBO.xml, MQSIv2HeaderBO.xml, MQSIv2MessageBO.xml.
2. Locate the adapter type: MQSeriesIntegratorV2AdapterType.xml.
3. Locate the adapter implementation: MQSeriesIntegratorAdapterV2JavaImp.xml.
4. Locate the adapter instance: MQSeriesIntegratorV2Adapter.xml.

On this platform:	The files are located in:
Windows NT	<partner_root>\com\extricity\adapters\ibm\mqseries\mqsi2 \test

Configuring Adapter for MQSeries Integrator (RFH2)

After you install Adapter for MQSeries Integrator (RFH2), you can configure it and take advantage of its default behavior, or you can modify the adapter to suit your specific needs (see *Modifying Adapter for MQSeries Integrator (RFH2)* on page 60).

Before you can customize the adapter, you must import all the required objects.

Importing the business objects

You need to import the business objects provided with the adapter.

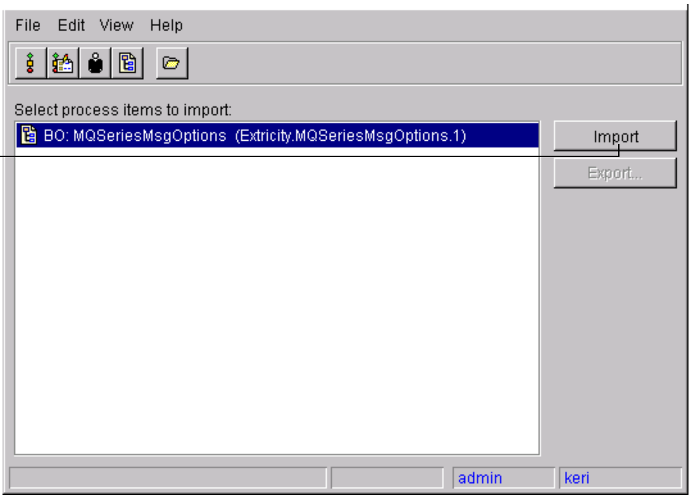
1 In the Partner Agreement Manager window, choose Import/Export Manager from the Tools menu.

2 In the Import/Export Manager, choose Open for Import from the File menu.

3 Browse to find the MQSeriesMsgOptionsBO.xml business object. Then click Open.

On this platform:	The files are located in:
Windows NT	<partner_root>\com\extricity\adapters\ibm\mqseries\mqsi2 \test

The business object appears in the list for importing.



BO: MQSeriesMsgOptions (Extricity.MQSeriesMsgOptions.1)

Click the Import button to add this business object to your PAM business objects.

4 Click Import. An Operation Complete dialog indicates success.

5 The MQSeriesMessageOptions business object and element definition set appear in your Business Objects folder.

6 Repeat steps 2 through 4 for the MQSIv2HeaderBO.xml business object.

7 The MQSIv2Header business object and element definition set appear in your Business Objects folder.

8 Repeat steps 2 through 4 for the MQSIv2MessageBO.xml business object.

9 The MQSIv2Message business object and element definition set appear in your Business Objects folder.

Importing the Adapter Type

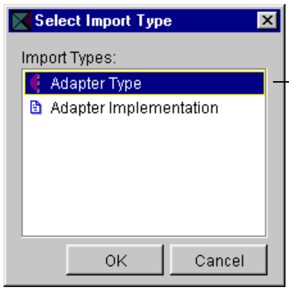
Next, you need to import the adapter type.

1 Start the Adapter Manager.

2 Choose Adapter Designer from the Tools menu to open the Adapter Designer.

3 In the Adapter Designer, choose Import from the File menu.

The Select Import Type dialog box appears.



You can import either an adapter type and an adapter implementation.

4 Select Adapter Type and click OK. In the Import dialog box, browse to find the MQSeriesIntegratorV2AdapterType.xml file.

On this platform:	The files are located in:
Windows NT	<partner_root>\com\extricity\adapters\ibm\mqseries\mqsi2 \test

5 Click Open. The adapter type appears in your Adapter Types folder.

Note: If you have previously installed an MQSeriesIntegratorV2 Adapter Type, the import will not work. If you are reinstalling, you must delete the previous instance; then you can delete the implementation and type.

Importing the Adapter Implementation

Next, you need to import the adapter implementation.

1 In the Adapter Designer, choose Import from the File menu.

2 Select Adapter Implementation from the list and click OK.

3 Browse to find the MQSeriesIntegratorAdapterV2JavaImp.xml adapter implementation. Then click Open.

On this platform:	The files are located in:
Windows NT	<partner_root>\com\extricity\adapters\ibm\mqseries\mqsi2 \test

4 The adapter implementation appears under the adapter type.

Importing the adapter instance

Finally, you need to import the adapter instance.

1 In the Adapter Manager window, choose Import Adapter Instance from the Server menu.

2 Browse to find the MQSeriesIntegratorV2Adapter.xml adapter instance. Then click Open.

On this platform:	The files are located in:
Windows NT	<partner_root>\com\extricity\adapters\ibm\mqseries\mqsi2 \test

3 The adapter instance appears in the Adapter Instances list.

Working with Adapter for MQSeries Integrator (RFH2)

Here's a brief overview of the steps you follow when working with Adapter for MQSeries Integrator (RFH2):

Step 1 Install Adapter for MQSeries Integrator (RFH2) and update the CLASSPATH with the Java classes.

Step 2 Verify the installation. Make sure that the basic adapter example code works and the adapter starts up. Create simple Get and Post operations and test a simple private process. Test an event with a simple public process. Check the status BO for results.

Step 3 Plan your Adapter for MQSeries Integrator (RFH2), including its required operations, events, and queues.

Step 4 Define and create the necessary business objects in PAM (if they do not already exist).

Step 5 Copy and create your own Java class files (for example, `com.<your_company>.adapters.ibm.mqseries.mqsi2.MQSeriesIntegratorAdapter`).

Step 6 Duplicate the adapter type.

Step 7 Add new adapter instance properties if required.

Step 8 Modify the adapter instance CustomClass property to point to your own Java class files.

Step 9 Go to the adapter operations and change the business objects from the example default to your own business object names.

Step 10 Modify your copy of the MQSeriesIntegratorCustomExample as required (for example, instead of Example_Inventory_Item BO, your own business object).

Step 11 Test your work as you build up the adapter.

Step 12 Add and/or customize events as required.

Step 13 Enhance the adapter with more advanced options after you work through the basics (for example, adding new operations, advanced mapping techniques, specialized events).

Modifying Adapter for MQSeries Integrator (RFH2)

Although you can use Adapter for MQSeries Integrator (RFH2) as is, the default behavior and settings might not meet your business needs. For example, business objects used in the adapter can be changed to something more appropriate for your business needs. Other customizations can vary from simply changing an operation name to using the API to make major changes to the adapter.

WARNING: When you start customizing the adapter, it is a good idea to do so in your own work space, for example:
`<partner_root>\com\`**`<your_company>`**`\adapters\ibm\mqseries\mqsi2\test`
This helps ensure that any future Adapter for MQSeries Integrator (RFH2) updates do not overwrite your custom work.

IMPORTANT: See the *Partner Agreement Manager Adapter Developer's Guide* for more information about customizing adapters.

The following are tips to bear in mind during implementation:

- Customize the operations in the adapter type to make it easier for process designers to select operations correctly.

  The PAM process builders who will use this adapter in private process extension actions rely on the names of the operations and their inputs and outputs to identify the correct operation. The operations provided with the Adapter for MQSeries Integrator (RFH2) type are intended as templates for your own specialized operations. Therefore, it's a good idea to change the generic PostMQSI operation name to something more meaningful, such as PostPurchaseOrder. It's also a good idea to remove operations that you don't expect process designers to use. For example, if you do not expect to use unparsed messages, you might want to remove the related operations from your adapter type.

- Review template operations provided in Adapter for MQSeries Integrator (RFH2) and add new operations as necessary.

  The template operations provided with Adapter for MQSeries Integrator (RFH2) represent only a portion of the full range of operations you can implement using the MQSeriesAdapterCore class public methods combined with MQRFH2 class methods. For example, although the template operations call the MQSeriesAdapterCore class getMessage and/or sendMessage functions only once (if at all) and create generic MQRFH2 instances, you can include operations that call these functions more than one time within an operation. This allows you to encapsulate more complex messaging interactions within a single operation, such as a single PAM business object to multiple MQSeries messages.

For example, a PAM process might include a purchase order business object that corresponds to several MQSeries messages—one for the header, plus additional messages for each individual order line. The easiest way to implement this would be to add a new basic operation—GetPurchaseOrder—that calls getMessage multiple times in unparsed mode (that is, using variants instead of business objects) and copy information from the resulting variants into your Purchase Order business object.

Note: Before you begin writing custom Java code for your adapter that deals with messages to/from MQSI, you should have a solid understanding of the formats defined in MQSI, which applications require those formats, and how the message broker is mapping messages and routing them to their destinations. Information you need to have before coding includes message formats, queues, queue managers, channels, listeners, and the MQRFH2 header format.

To configure Adapter for MQSeries Integrator (RFH2), implement any custom code by writing one or more MQSeriesAdapterCustom classes or subclasses.

For more information about using or implementing the MQSeriesAdapterCore, MQSeriesIntegratorRFH2, and MQSeriesAdapterCustom classes, see the accompanying Javadoc reference. You might also want to review the MQSeriesIntegratorCustomExample subclass, which illustrates how to implement the various custom methods contained in the MQSeriesAdapterCustom class.

Tip: See <product root>\Docs\Javadoc\MQSI\index.html for the Javadoc on NT.

Testing Adapter for MQSeries Integrator (RFH2)

As a minimum, an adapter instance must meet these criteria before you can use it in an extension action. You must be able to:

- start up and shut down the adapter instance without error.
- verify that you can connect to the queue manager.

Before you release an adapter for general use by process designers, it's a good idea to test any custom code you have implemented. The easiest way to test custom code is to create a sample Partner Agreement Manager private process that uses extension actions that execute the adapter operations.

To test Adapter for MQSeries Integrator (RFH2):

1. Create a PAM private process that includes attempts to post and get a series of business objects. Make sure that you execute every path in your custom code.
2. After getting a business object, make sure—in VBScript (NT) or JavaScript (NT and UNIX)—that the contents are as you expect.

   Tip: Use the StatusBO operation output and check the results.
3. Review the log file to make sure that no exceptions were reported within the adapter itself.
4. While debugging, try setting the adapter's DebugMessaging property to Terse or Verbose.
5. Turn on MQSeries tracing and look at the MQSeriesTrace.log file (located in the partner root directory) in the event of any MQSeries exceptions.
6. Test the public process events.

Adapter for MQSeries Integrator (RFH2) reference

This section describes in more detail the sample Adapter for MQSeries Integrator (RFH2) type that comes packaged with the product. The class names, properties, operations, and events are described.

Note: The operations are intended as templates only. You are encouraged to rename these operations and their parameters as needed, or design your own operations from scratch that use the MQSeriesAdapterCore class's methods in other ways.

Finally, since this sample adapter is a subclass of the underlying PAM Adapter for MQSeries Messaging, the messaging adapter's operations are also included.

Note: Each operation has a StatusBO that returns either result="success" or result="failure". If the operation fails, the StatusBO also returns the specific reason why it failed.

Adapter for MQSeries Integrator (RFH2) type

These are the components of the Adapter for MQSeries Integrator (RFH2) type.

Class name	com.extricity.adapters.ibm.MQSeries.mqsi2.MQSeriesIntegrator Adapter
Description	The MQSeriesIntegratorAdapter class is a template for MQSeries adapter implementations that use MQSeriesIntegrator version 2.0. While this can be run as is, you might need to make site-specific customizations, particularly for event polling. This class requires the presence of the MQSeriesAdapterCore, MQSeriesIntegratorRFH2 and MQCustomIntegrator classes. This template implementation works with the adapter type that comes with the MQSeries Integrator v2. If changes are made here, then corresponding changes might need to be made in the type. This class represents the adapter implementation.
Class name	com.extricity.adapters.ibm.MQSeries.mqsi2.MQSeries IntegratorRFH2
Description	This class provides several helper methods for manipulating an MQ rules and formatting header. This class represents a root class for accessing the common fields in the header. **Note**: Publish/Subscribe is not supported with the MQRFH2 header format.
Class name	com.extricity.adapters.ibm.MQSeries.mqsi.MQSeriesIntegrator CustomExample
Description	This class contains stub methods that can be implemented on site. These methods are as follows: ■ sendReply. Send a low-level acknowledgment to an incoming message. **Note:** You must customize this method (or nothing will happen). ■ receiveReply. Await a low-level acknowledgment to an outgoing message. **Note:** You must customize this method (or nothing will happen). ■ callInMap. Custom data transformation method for incoming messages; called whenever a Mapped or MQSI operation is indicated and the IncomingMap property is left null. ■ callOutMap. Custom data transformation method for outgoing messages; called whenever a Mapped or MQSI operation is indicated and the OutgoingMap property is left null. The default behavior of the mapping methods is to convert business objects to/from XML as well as read/write default MQRFH2 headers.

Properties	Channel	Name of server connection channel by which to connect to queue manager. **Mandatory**.

CorrelationID	The value is used only during event checking (that is, checkForEvents); otherwise, it is ignored. **Optional**.
CustomClass	Name of example Custom class. Must be the fully qualified class pathname, dot-delimited, relative to the classpath. Defaults to com.extricity.adapters.ibm.mqseries.MQSeries IntegratorCustomExample. **Mandatory**.
DebugMessaging	One of three values: Off, Terse, Verbose, representing different levels of debug messaging. Messages are written to the Adapter Server console. Debug messaging might be useful during development and deployment, but is generally set to Off in a production environment. **Mandatory**.
DefaultQueue	Name of a default queue to use for sending and receiving messages. Can be overridden in code. **Optional**.
Hostname	Name of host on which QueueManager resides. This can be an IP address or the IP host name. **Mandatory**.
IncomingMap	Command line that must be executed in order to map incoming messages. **Optional**. If left blank, the adapter calls the callInMap custom method instead for all incoming mapped messages. If populated, then mapped messages undergo the following steps: 1. Write incoming MQSeries message contents to a file, <MapWorkingDirectory>\inMsg.txt (Windows NT) 2. Call the property value as a system command, which translates the MQSeries file to an XML file, <MapWorkingDirectory>\outBO.xml (Windows NT) 3. Read in the XML file and fill a business object with the contents. **NOTE:** If this mapping feature is used for MQSI adapters, the map executable must be MQRFH2-aware.

MQSeriesTraceLevel	Sets the MQSeries trace level. One of six values: Off, or one of the five levels of detail that the MQSeries trace facility records debug-level information to the file MQSeriesTrace.log, located in the partner root directory. **Mandatory**.
MapWorking Directory	The location in the local file system to use for writing and reading files during a command-line mapping execution. Ignored if IncomingMap and OutgoingMap are blank. **Optional**.
OutgoingMap	Command line that must be executed in order to map outgoing messages. **Optional.** If left blank, the adapter calls the callOutMap custom method instead for all outgoing mapped messages. If populated, mapped messages undergo the following steps: 1. Write BO to an XML file, <MapWorkingDirectory>\inBO.xml (Windows NT) 2. Call the property value as a system command, which translates the XML file to an MQSeries message file, <MapWorkingDirectory>\outMsg.txt (Windows NT) 3. Read in the MQSeries file and put contents into an MQSeries message. **Note:** If this mapping feature is used for MQSI adapters, the map executable must be MQRFH2-aware.
Password	The password associated with the username. **Optional**, but must be entered if security is enabled on the server connection channel.
Port	Name of port to which channel is listening. Defaults to 1414. **Mandatory**.
QueueManager	Name of queue manager to connect to. **Mandatory**.
UserId	The username used to connect to the queue manager. **Optional**, but must be entered if security is enabled on the server connection channel.

	WaitInterval	The length of time, in milliseconds, that an attempt to Get a message from a queue must wait before erroring out, if a message is not immediately available. **Mandatory**. Can be overridden in code.
Operations	**BrowseMapped**	Retrieves a message, calls a map, and returns the business object. Does not consume message in the queue. **NOTE:** If the IncomingMap property is specified, the operation writes the incoming message to a file, calls the command specified in this property, reads in the resulting XML, and returns the BO. If this property is left null, calls the Custom callInMap method (stub function).
	Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
	Output	Business object. Also, status BO. Defaults to the Example_Inventory_Item BO. Modify this to output to your own BO as necessary.
	BrowseMQSI	Retrieves a message without consuming it in the queue, calls a map, including MQRFH2, and returns the functional BO. **NOTE:** If the IncomingMap property is specified, the operation writes the incoming message to a file, calls the command specified in this property, reads in the resulting XML, and returns the BO. If this property is left null, then calls the custom map method.
	Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
	Output	The MQSIv2Message BO specified in the adapter type definition. Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
	BrowseUnparsed	Retrieves a message and returns raw contents to the private process. Message is not consumed in the queue.
	Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.

Output	Variant, representing message contents. Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
GetMapped	Retrieves a message, calls a map, and returns the business object. Consumes the message in the queue. NOTE: If the IncomingMap property is specified, the operation writes the incoming message to a file, calls the command specified in this property, reads in the resulting XML, and returns the BO. If this property is left null, calls the Custom callInMap method (stub function).
Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Business object. Also, status BO. Defaults to the Example_Inventory_Item BO. Modify this to output to your own BO as necessary.
GetMQSI	Retrieves a message, calls a map, including MQRFH2, and returns the functional BO. Consumes the message in the queue. NOTE: If the IncomingMap property is specified, the operation writes the incoming message to a file, calls the command specified in this property, reads in the resulting XML, and returns the BO. If this property is left null, then calls the custom map method.
Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	The MQSIv2Message BO specified in the adapter type definition. Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
GetUnparsed	Retrieves a message and returns raw contents to the private process. Message is consumed in the queue.
Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.

Output	Variant, representing message contents. Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
PostMapped	Send a message using map for data transformation. **Note:** If the OutgoingMap property is specified, the operation writes the BO as an XML file, calls the command specified in this property, reads the resulting bytes into a new MQMessage, and sends it. If this property is left null, calls the Custom callOutMap method (stub function).
Input	Business object, which defaults to the Example_Inventory_Item BO. Modify this to output to your own BO as necessary. MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
PostMQSI	Send a message using map for data transformation, with MQRFH2 header included. **Note:** If the OutgoingMap property is specified, the operation writes the BO as an XML file, calls the command specified in this property, reads the resulting bytes into a new MQMessage, and sends it. If this property is left null, then calls the custom map method.
Input	The BO specified in the adapter type definition. Defaults to the Example_Inventory_Item BO. Modify this to output to your own BO as necessary. The MQSIv2Header BO specified in the adapter type definition. MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.

PostUnparsed	Send an unparsed, unmapped message.
Input	Variant, representing the outgoing message contents. MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
RequestMapped	Send a message (mapped), then wait for a return message (mapped).
Input	Business object, which defaults to the Example_Inventory_Item BO. This BO corresponds to the output message (map is applied to this BO as in PostMapped). Specified in the adapter type definition. Modify this to output to your own BO as necessary. A variant, which must be specified by the private process, indicating the queue name where the reply must be sent. MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Business object, which defaults to the Example_Inventory_Item BO. This BO corresponds to reply message (map is applied to this BO as in GetMapped). Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
RequestMQSI	Send a message (mapped), with MQRFH2, then wait for a return message (mapped) with MQRFH2.

	Input	Business object, which defaults to the Example_Inventory_Item BO. This BO corresponds to the output message (map is applied to this BO as in PostMapped). Specified in the adapter type definition. Modify this to output to your own BO as necessary.
		A variant, which must be specified by the private process, indicating the queue name where the reply must be sent.
		The MQSIv2Header BO specified in the adapter type definition.
		MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
	Output	The MQSIv2Message BO specified in the adapter type definition (map is applied to this BO as in GetMapped). Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
	RequestUnparsed	Send a message (unparsed), then wait for a return message (unparsed).
	Input	Variant, representing contents of the outgoing message.
		A variant, which must be specified by the private process, indicating the queue name where the reply must be sent.
		MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
	Output	Variant, representing contents of the reply message. Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
Events	MQEvent. Contains variant data as the default. Functionality of checkForEvents is to poll a given queue and return any messages that appear, as in the GetUnparsed operation.	
	You can use a business object (BO) instead of a variant for your event. You can also create your own event. You need to modify the checkForEvents method in MQSeriesIntegratorAdapter.java as well.	
	If the business object (BO) being returned is an MQSIv2Message, then the adapter does not need to be customized. If you are using a BO that doesn't contain the MQRFH2 header and contains only the data, then the adapter might need to be customized.	

Tip: Each operation has an optional StatusBO that returns either result="success" or result="failure". If the operation fails, the StatusBO also returns the specific reason why it failed. It is a good idea to check this status.

Adapter for MQSeries Integrator (RFH2) business objects

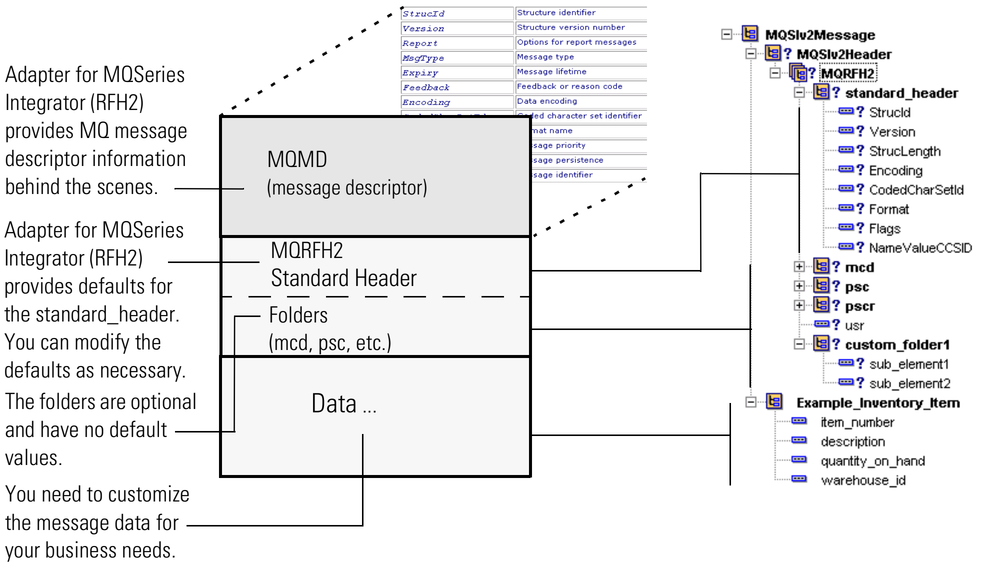
The following table provides details about Adapter for MQSeries Integrator (RFH2) business objects.

Business Object	MQSeriesMsgOptions	
Description	This business object contains message-level options for queue name and correlation ID. It can be used with all the predefined operations.	
Elements	QueueName	Queue name, which is used as required for various operations (for example, as input for GetMQSI operation—this is the queue where the GetMQSI operation *gets* the message).
	CorrelationID	CorrelationID, which is used as required for operations to relate messages to the application. See IBM MQSeries documentation for more information.
Business Object	MQSIv2Header	
Description	This business object corresponds to the IBM MQSeries Integrator MQRFH2 header. It can be used with the MQSI operations (for example, RequestMQSI). See the *IBM MQSeries Integrator Programming Guide* for more information.	
Elements	standard_header	The fixed part of the MQRFH2 header. All fields are optional. If not defined, the defaults are used. (See the *IBM MQSeries Integrator Programming Guide*, chapter 4, for the initial value defaults.) By default, only the standard_header is populated/ defined.
	mcd	Message content descriptor, which contains elements to describe the structure of the message data (for example, XML or PDF).
	psc	Publish/subscribe command, describing command messages to the broker. **Note:** The publish/subscribe features are not supported with this adapter release.

	pscr	Contains information in response to publish/subscribe messages. **Note:** The publish/subscribe features are not supported with this adapter release.
	usr	Application (that is, user) defined properties. With a string representation of XML, you can add a comment or other descriptive information to your message headers. See *Using the usr folder* on page 75 for more information.
	other	Optionally, you can add your own group elements. These elements can be used to include metadata for advanced processing requirements, brokering, or data transformation. See *Creating other (custom) header folders* on page 76 for more information.
Business Object	MQSIv2Message	
Description	The Adapter for MQSeries Integrator (RFH2) message. It consists of two parts: ■ A container for the header of the message. - If the message received contains an MQRFH2 header, this container gets populated with the header information. If not, then this is left empty. ■ The body of the message (by default, it is set to Example_Inventory_Item). Modify this to suit your business needs.	
Elements	MQSIv2Header	See above.
	Example_Inventory _Item	Sample message data BO. This must be customized (that is, replace this BO with the one you need) to fit your needs.

Working with business objects

Adapter for MQSeries Integrator (RFH2) provides basic functionality when you install it. First, you must understand the structure of the messages being passed by the adapter.



You can make changes to the standard header, the folders, and the message data itself. See the *IBM MQSeries Integrator Programming Guide* for more information.

Modifying the MQSIv2Header

If your message header needs to include more than the default information, then you need to customize the adapter for this. For example, you can change the default standard_header elements, or you can describe the message contents using the mcd folder. You can also create your own custom folders, which are called “other” folders in the MQSeries documentation.

Using the standard_header

You can modify the standard_header defaults, if necessary. To do so, you must populate and bind the input business object to the operations, using a private process script action.

Using the mcd folder

You can use a private process script action to describe your message contents. If you want to use the mcd header folder, you must populate and bind the input business object to the operations.

Using the psc folder

If you want to change the defaults in the psc header folder, you must populate and bind the input business object to the operations, using a private process script action. See chapter 5 in the *IBM MQSeries Integrator Programming Guide* for more information.

Note: The publish/subscribe features are not supported with this adapter release.

Using the pscr folder

To change the defaults in the pscr header folder, you must populate and bind the input business object to the operations, using a private process script action. See chapter 5 in the *IBM MQSeries Integrator Programming Guide* for more information.

Note: The publish/subscribe features are not supported with this adapter release.

Using the usr folder

You can add your own application-defined properties with the usr header folder. This feature can be used to add a comment or other descriptive information to your message headers.

The following shows how the usr folder is populated with an XML string. This can be done in the BO Mapper Utility or the Script Editor. (This script would be executed before the operation call.)

```
function main() {
    createBO("mqrfh");
    var mqrfh2 =
mqrfh.getElementSequence("MQRFH2").newElement();
    ..... // populate all the standard folders as required.
    mqrfh2.setData("usr", "<usr>This is a test string</usr>");
```

```
-OR-
```

```
    mqrfh2.setData("usr", "<usr> <user_defined_xml_element>test
data</user_defined_xml_element></usr>");
.
.
.
}
```

NOTE: PAM business objects cannot handle the greater than (>) or less than (<) characters (as part of the data within an element). So a "<" is translated to "&lt" and a ">" is translated as "&gt".
For example, for the case where usr is set to:
`<usr>This is a test string</usr>`
The result is:
`&lt;usr&gt;This is a test string&lt;/usr&gt;`

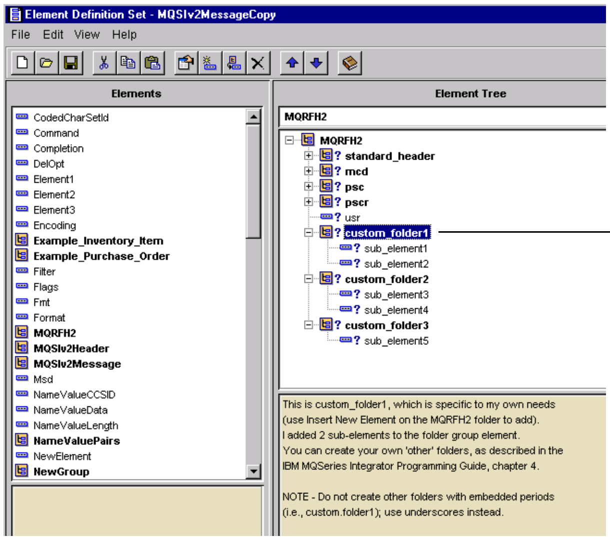
CREATING OTHER (CUSTOM) HEADER FOLDERS

If you want to create your own group elements, you can create customized "other" header folders. These user-defined elements can be used to include metadata for advanced processing, brokering, or data transformation.

Here are the steps to follow when creating a customized folder:

1. Open the business object element definition set MQSIv2Header. Use File > Save As to create a copy in your partner folder. (You can modify your own partner version; you cannot modify the original.)
2. Open your copy of the MQSIv2Header element definition set. From the MQRFH2 element/group, use Insert New Element to add a new other folder (element/group). Add elements to the new other folder as required.
3. Create a business object from the new element definition set. Freeze the business object.
4. In the Adapter Designer, modify the adapter operations to use the new business object. See the *Partner Agreement Manager Adapter Developer's Guide* for more information.
5. Modify the MQSeriesIntegratorAdapter.java file.
6. Recompile the MQSeriesIntegratorAdapter.java file.
7. Restart the Adapter Server.
8. Start the adapter instance.
9. Create the process to use the new adapter instance.

To make changes to the business objects and element definition sets, you need to make your own copies of the adapter-provided ones. Then, you can add the folder(s) and element(s) you need. As an illustration:



CodedCharSetId
Command
Completion
DelOpt
Element1
Element2
Element3
Encoding
Example_Inventory_Item
Example_Purchase_Order
Filter
Flags
Fmt
Format
MQRFH2
MQSIv2Header
MQSIv2Message
Msd
NameValueCCSID
NameValueData
NameValueLength
NameValuePairs
NewElement
NewGroup

MQRFH2

- **MQRFH2**
  - **? standard_header**
  - **? mcd**
  - **? psc**
  - **? pscr**
  - ? usr
  - **? custom_folder1**
    - ? sub_element1
    - ? sub_element2
  - **? custom_folder2**
    - ? sub_element3
    - ? sub_element4
  - **? custom_folder3**
    - ? sub_element5

This is custom_folder1, which is specific to my own needs (use Insert New Element on the MQRFH2 folder to add).
I added 2 sub-elements to the folder group element.
You can create your own 'other' folders, as described in the IBM MQSeries Integrator Programming Guide, chapter 4.

NOTE - Do not create other folders with embedded periods (i.e., custom.folder1); use underscores instead.

Use Insert New Element to add your own other (custom) folder(s) and element(s).

You also need to update the MQSeriesIntegratorAdapter.java file to include the new other folder name(s). The following example illustrates this:

```
/**
* List of all "other" folders defined in the MQRFH2 header.
* Initially this list is empty, but this list will have to be
* customized if there are folders added to the MQRFH2 header
structure
* other than the "mcd", "psc", "pscr" and "usr".
* This has to be public static.</p>
*/
public static String[] other_folder_list = {
"custom_folder1",
"custom_folder2",
"custom_folder3"
};
```

Tips for other (custom) folders

- It is safer to add *optional* elements and sub-elements to the element definition set. This way, unused fields do not render the BO invalid.
- See the *IBM MQSeries Integrator Programming Guide*, chapter 4, for guidelines on creating and using the other (custom) folders.

- MQSeries allows "custom.folder"—PAM does not allow embedded periods—use "custom_ folder" (with an underscore) as an alternative.
- Only the top-level folders need to be defined in the Java code (that is, the same level as the "psc" and "mcd" folders). Lower-level folders and sub-elements do not need to be defined.

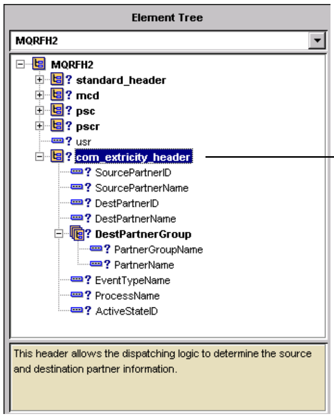
Working with other (custom) header folders

When an incoming message is intended to start a public process, you can use an other (custom) header folder and customized dispatching logic (that is, logic to determine which PAM event a given message corresponds to) to determine information such as:

- which event type or subprocess the incoming message needs to be associated with.
- if the process contains partner groups, which partners need to be notified.
- if the message originated from another partner, the identity of that partner.

If this information is provided (by the originating application) in a well-defined header, then the dispatching logic does not need to scan the message contents or other fields. Instead, the message processing logic can be determined from the header alone.

The following element definition set illustrates an other (custom) header:



This header allows the dispatching logic to determine the source and destination partner information.

One possible use for an other (custom) folder is to include B2B partner information.

Modifying the MQSIv2Message

If your message data does not include the default Example_Inventory_Item business object, then you need to customize the adapter to fit your needs.

Here are the steps that you follow to modify your message (data) business-object-related information:

1. Open the business object element definition set **MQSIv2Message**. Use File > Save As to create a copy in your partner folder. (You can modify your own partner version; you cannot modify the original.)
2. Create a business object for your message data, if one does not already exist. This replaces the Example_Inventory_Item BO in your copy of MQSIv2Message. (This is the data to be returned from the Get operations.)
3. Open your copy of the **MQSIv2Message** element definition set. Replace the Example_Inventory_Item element with your new BO.
4. Create a business object from the new element definition set. Freeze the business object.
5. Modify the MQSeriesIntegratorAdapterCustomExample.java file.
6. Recompile the MQSeriesIntegratorAdapterCustomExample.java file.
7. Use the new custom class in the adapter instance property.

The following example uses the Example_Inventory_Item to construct the messageBO (callInMap method). There are five arguments that you need to customize in the code. These can be found in the call to Utilities.constructMessageBO:

```
constructMessageBO(getBO,  // the BO being returned
          MQRFH2, // The MQSeriesIntegrator RFH2 // object
          EXAMPLE_INVENTORY_ITEM, // BO type name
          EXAMPLE_INVENTORY_ITEM, // BO definition name
          1,                      // version of the BO
          EXTRICITY,    // Owner of the output BO
          3L,           // Partner ID of owner of BO
          sr);          // String reader
```

The third through seventh arguments need to be modified.

For this argument:	You need to provide this information:
BO type name	The type name of the new business object that you created.
BO definition name	The new business object definition name.
Version of the BO	The version number of the new business object.
Owner of the output BO	The business object owner’s partner name.
Partner ID of BO owner	The business object owner’s partner ID.

Do not modify the other arguments.

CHAPTER

6

Using Adapter for MQSeries Publish/Subscribe

Read this chapter to learn how configure and use WebSphere® Partner Agreement Manager Adapter for MQSeries® Publish/Subscribe version 2.1.

This chapter includes the following sections:

Adapter for MQSeries Publish/Subscribe combines the message distribution and subscription features of IBM MQSeries Publish/Subscribe with the business-to-business integration capabilities of WebSphere Partner Agreement Manager.

Note: Partner Agreement Connect (PAC) is a version of Partner Agreement Manager (PAM) designed for partners who do not initiate processes. Because the same core documentation set is used for both PAM and PAC, the documentation uses the term “Partner Agreement Manager” or “PAM” to refer to both products.

About IBM MQSeries Publish/Subscribe

IBM MQSeries Publish/Subscribe is freely available from IBM as a SupportPac™ to MQSeries 5.x.

When you use the IBM MQSeries Publish/Subscribe capability, a broker listens to specialized queues on a given queue manager. Applications needing to exchange messages via a publish/subscribe protocol send messages to a control queue. The control queue, in turn, notifies the broker that they are to be publishers and subscribers of specific message topics. Then, published messages are sent to specific queues (known as streams). From there, the broker forwards them to queues associated with the subscribers of the given message’s topics. Both control and published messages must include an MQSeries Rules and Format Header (MQRFH) that identifies the nature of the message. This MQRFH is a variation of the IBM MQSeries Integrator MQRFH.

About Adapter for MQSeries Publish/Subscribe

Adapter for MQSeries Publish/Subscribe, used in conjunction with Adapter for MQSeries Messaging, provides further support to adapter developers who must work with MQSeries messages that are published or subscribed via the publish/subscribe broker. It includes support for managing the publish/subscribe MQRFH, as well as template adapter operations and sample custom code that illustrates how to send control messages, send publications, receive subscribed messages, and manipulate the MQRFH of each message, all via an intuitive API.

This version of Adapter for MQSeries Publish/Subscribe is compatible with WebSphere Partner Agreement Manager (PAM) version 2.1. It requires the IBM MQSeries classes for Java (MQ base Java)™, which is compatible with all versions of MQSeries through version 5.1.

Adapter for MQSeries Publish/Subscribe consists of:

- class files corresponding to all Java classes.
- Java files for MQSeriesPubSubAdapter class.
- Java and class files for an example MQSeriesAdapterCustom subclass (MQSeriesPubSubCustomExample.java and MQSeriesPubSubCustomExample.class).
- XML representations of template adapter type and implementation (MQSeriesPubSubAdapterType.xml, MQSeriesPubSubAdapterJavaImp.xml).
- Javadoc reference for all class files.

Key features of Adapter for MQSeries Publish/Subscribe include:

- *Adapter for MQSeries Publish/Subscribe implementations are subclassed from the base Adapter for MQSeries Messaging.* Other than managing the publish/subscribe MQRFH, in all other respects the same code can be reused.
- *Customizable classes allow you to manage the publish/subscribe MQRFH.*
- *Sample publish and broadcast-reply operations* automatically call the MQRFH class methods for you.
- *The adapter provides the ability to construct and send control messages*, including publisher and subscriber registration and deregistration, and reloading and deleting retained messages. These control messages can be sent from any adapter operation.
- *The ability to receive broker-generated messages as low-level replies,* via a subclass of the base MQSeries adapter's custom class is provided. If these messages include error conditions, then the adapter can automatically generate an exception containing the appropriate error information.
- *The adapter provides multiple subscribers per adapter instance* as a checkForEvents feature. The adapter keeps track of which business objects are associated with which message topic, and which data transformations to apply.

About the Adapter for MQSeries Publish/Subscribe environment

See *Installing Adapters for MQSeries* on page 5 for general system requirements. In addition to general requirements, Adapter for MQSeries Publish/Subscribe requires the following software:

- The PAM Adapter for MQSeries Messaging (see *Using Adapter for MQSeries Messaging* on page 9).
- The IBM MQSeries publish/subscribe broker (installed and configured on an MQSeries queue manager).
- IBM MQSeries classes for Java (MQ base Java), which is available separately from IBM

Important: If you are using MQSeries 5.1, make sure that the latest MQSeries CSD*nn* (Corrective Service Diskette) is installed before using the publish/subscribe broker.

See the *IBM MQSeries Publish/Subscribe* documentation for information on configuring a publish/subscribe broker.

Updating the CLASSPATH

Adapter for MQSeries Publish/Subscribe requires that the IBM MQSeries classes for Java (MQ base Java) be installed on the computer where the Adapter Server is running. This software is separately available from IBM.

Note: The IBM SupportPac, MA88: MQSeries classes for Java and MQSeries classes for Java Message Service, contains the required MQSeries software.

The following jar files must be installed (on both the PAM client and server computers):

- MQSeries\java\lib\com.ibm.mq.jar
- MQSeries\java\lib\com.ibm.mqbind.jar
- MQSeries\java\lib\com.ibm.mq.iiop.jar

For more information on IBM MQSeries Publish/Subscribe products and software, see the IBM web site:

http://www.software.ibm.com/ts/mqseries/support

After you install the IBM MQSeries classes for Java (MQ base Java), you must update your system CLASSPATH.

To update your CLASSPATH on Windows NT:

1. From the Windows Start menu, choose Settings > Control Panel > System.
2. Click the Environment tab.
3. Select the CLASSPATH system variable.

   The current CLASSPATH value appears.
4. Edit this field as necessary to include the class paths.

   For example, if the Java client components are installed in C:\mqm\java\lib, your CLASSPATH should include C:\mqm\java\lib\com.ibm.mq.jar, C:\mqm\java\lib\com.ibm.mqbind.jar, and C:\mqm\java\lib\com.ibm.mq.iiop.jar.
5. Click Set, and then click Apply.
6. Restart the Adapter Server so the new MQSeries Java classes can be recognized.

About configuring IBM MQSeries Publish/Subscribe

Adapter for MQSeries Publish/Subscribe is designed to require minimal setup in MQSeries.

- An MQSeries adapter relies on any queue managers it accesses to be up and running, with all channels and queues properly defined. The queue manager must be accessible via the MQSeries Java Client, it must have an MQSeries publish/subscribe broker running against it, and the publish/subscribe-specific queues (control, publish, reply, subscription) must be defined correctly.

Important: Be sure to edit the MQSeriesPubSubCustomExample.java file to specify the replyToQueueManagerName and replyToQueueName variables (set them to your local queue manager).

The adapter reports an exception if you attempt to communicate with a queue manager that is down, with an undefined or full queue, or with a disabled or non-existent channel. The adapter doesn't enable or actively troubleshoot an MQSeries network—beyond reporting these types of exceptions.

If the publish/subscribe broker is inoperative, sending messages to it results in those messages queueing up without any feedback. Attempts to receive messages fail in this case. In general, Adapter for MQSeries Publish/Subscribe does not actively troubleshoot an inoperative or otherwise improperly functioning MQSeries queue manager or MQSeries publish/subscribe broker, beyond reporting exceptions and time-outs encountered.

- When the adapter encounters an exception condition in MQSeries, the MQSeriesAdapterException contains the exact exception text returned by MQSeries.

  This exception text normally contains an error code number and short description. For more information on possible error conditions, see the *IBM MQSeries Messages* reference.

Installing Adapter for MQSeries Publish/Subscribe

You must install Adapter for MQSeries Publish/Subscribe—on the same computer where the Adapter Server is running—by copying the appropriate java and class files into your PAM partner directory.

Before you install Adapter for MQSeries Publish/Subscribe, you must install and configure the MQSeries queue manager. See *About IBM MQSeries Publish/Subscribe* on page 82 for more information.

Although you can run both the MQSeries queue manager and the Adapter Server on the same computer, we strongly recommend that you install them on different computers. The two computers must be able to communicate via TCP/IP.

Make sure that:

- the MQSeries publish/subscribe broker is running against the correct queue manager, that the broker is configured, and that broker-specific queues are set up. By default, the adapter sends messages intended for the broker to the SYSTEM.BROKER.CONTROL.QUEUE, and receives messages from MQSeries publish/subscribe in queues named PUBLISHER.REPLY.QUEUE and SUBSCRIBER.REPLY.QUEUE. The adapter sends publications to PUBLISHER.STREAM, and subscribed messages appear in SUBSCRIBER.QUEUE. Make sure that the PUBLISHER.STREAM is not sharable.
- Adapter for MQSeries Messaging is installed. Its files appear in this location: `<partner_root>\com\extricity\adapters\ibm\mqseries`.
- the IBM MQSeries classes for Java (MQ base Java) are installed on the computer where the Adapter Server is running, and make sure that the Adapter Server's CLASSPATH is set to include the Java Client class files (see *Updating the CLASSPATH* on page 84).

IMPORTANT: Make sure you *back up* your partner root directory *before* installing the adapter. This is especially important if you have ever used (and customized) previous versions of MQSeries adapters.

To install Adapter for MQSeries Publish/Subscribe:

1. Log on—as a user with administrative privileges—to the computer where the Adapter Server is installed.
2. Install Adapter for MQSeries Publish/Subscribe.

   On Windows NT:

   a. Copy MQSeriesPubSub2_1.zip to your partner root directory.

   b. Extract the zip file into your partner root directory.

IMPORTANT: If you previously installed an MQSeries adapter, a Confirm File Overwrite dialog appears. Click Yes to All (twice) to replace the existing files.

NOTE: If you have the PAM client and server on two different computers, you need to extract the files into the partner directory on both computers.

Adapter for MQSeries Publish/Subscribe files appear in this location:

On this platform:	The files are located in:
Windows NT	<partner_root>\com\extricity\adapters\ibm\mqseries\pubsub

Make sure that the java and class files are present in the same directory.

To import key elements:

For general importing guidelines, see *Importing key elements* on page 7.

▶ Move the file named MQSeriesMsgOptionsBO.xml from the Process Server computer to the computer where you're running the PAM client.

On this platform:	The files are located in:
Windows NT	<partner_root>\com\extricity\adapters\ibm\mqseries\pubsub \test

Configuring Adapter for MQSeries Publish/ Subscribe

After you install Adapter for MQSeries Publish/Subscribe, you can configure it and take advantage of its default behavior, or you can modify the adapter to suit your specific needs (see *Modifying Adapter for MQSeries Publish/ Subscribe* on page 91).

To configure Adapter for MQSeries Publish/Subscribe:

1 Implement any custom code by writing one or more MQSeriesAdapterCustom classes or subclasses.

See *Modifying Adapter for MQSeries Publish/Subscribe* on page 91 for tips on implementing custom code.

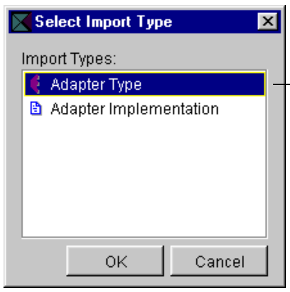
2 If the Adapter Server is not already running, start it.

3 Start the Partner Agreement Manager Process Server and the Adapter Server Administrator.

4 Choose Adapter Designer from the Tools menu to open the Adapter Designer.

5 In the Adapter Designer, choose Import from the File menu.

The Select Import Type dialog box appears.



You can import either an adapter type or an adapter implementation.

6 Select Adapter Type and click OK. In the Import dialog box, select the MQSeriesPubSub Adapter Type XML file located in: `<partner_root>\com\extricity\adapters\ibm\mqseries\pubsub\test`

This creates a template adapter type that you can edit to conform to your customized adapter code. For example, you can add any new operations you need.

- If you plan to use the publish/subscribe or mapped operations, you can edit their inputs and outputs to match the business objects you plan to get and post.
- Follow the instructions in the *Partner Agreement Manager Adapter Developer's Guide* to set default property values as appropriate for the adapter type.
- If you plan to subscribe to any publications, you must write a checkForEvents() method that is appropriate for your style of messaging. The sample checkForEvents() method included in MQSeriesPubSubAdapter.java sends a RequestUpdate message to the broker before checking for subscribed messages. If you are not using Retained Publications, don't send RequestUpdate messages.
- If you plan to subscribe to multiple message topics, each topic might require different data transformation logic. Make sure that your callInMap() method or your command-line maps know how to transform multiple message types to the correct business object types.
- If you plan to use subscribed message events (events containing business object data), create a corresponding event type.

7. In the Adapter Designer, choose Import again, select Adapter Implementation, and select the MQSeriesPubSub Adapter Java Imp XML file located in:
   `<partner_root>\com\extricity\adapters\ibm\mqseries\pubsub\test`

   This creates the adapter implementation. Make sure that the adapter implementation's class name and package names are correct.

8. In the Adapter Server Administrator, choose Adapter Manager from the Tools menu to start the Adapter Manager.

9. Choose Add from the Adapter menu to create a new instance of Adapter for MQSeries Publish/Subscribe, and follow the instructions in the *Partner Agreement Manager Adapter Developer's Guide* to set the connection property values.

   - If you plan to use a command-line-executable map for your Mapped operations, you must enter the system commands in the IncomingMap and OutgoingMap properties.
   - If your adapter checks for events, make sure that event polling is enabled and polls at an appropriate time interval.
   - If you have created a subclass of the MQSeriesPubSubAdapterCustom class for this instance, you must enter its name in the CustomClass property.
   - Set the MQSeriesTrace property according to your debugging needs. You might want to turn this on only during development, testing, or maintenance. The logs can quickly grow very large.
   - In the Adapter Manager, start the new adapter instance and resolve any exceptions that occur.

   As soon as the adapter instance has connected to the queue manager, you are ready to begin using it in extension actions.

It is unlikely that the adapter's default behavior will suit your business needs. The adapter instance communicates with only one queue. Also, remote applications receiving your MQSeries messages need to understand XML, and they also need to be able to construct the XML messages that your adapter consumes.

The following sections provide guidelines for customizing the adapter to suit your specific needs.

Modifying Adapter for MQSeries Publish/Subscribe

Although you can use Adapter for MQSeries Publish/Subscribe as is, the default behavior and settings might not meet your business needs. For example:

- The default mapping behavior for mapped operations is to convert business objects to/from XML with MQRFH.
- Publish and BroadcastReply operations send messages to a queue named PUBLISHER.STREAM and receive messages from a queue named SUBSCRIBER.QUEUE. Subscribed messages also arrive in SUBSCRIBER.QUEUE.
- Control messages are sent by most operations, as well as startup, to SYSTEM.BROKER.CONTROL.QUEUE.
- Broker reply messages are sent to PUBLISHER.REPLY.QUEUE and SUBSCRIBER.REPLY.QUEUE.
- Other operations connect to the queue specified in the DefaultQueue property.

As a result, you might need to customize Adapter for MQSeries Publish/Subscribe to suit your specific needs.

IMPORTANT: See the *Partner Agreement Manager Adapter Developer's Guide* for more information about customizing adapters.

The following are tips to bear in mind during implementation:

- Customize the operations in the adapter type to make it easier for process designers to select operations correctly.

  The PAM process builders who will use this adapter in private process extension actions rely on the names of the operations and their inputs and outputs to identify the correct operation. The operations provided with the Adapter for MQSeries Publish/Subscribe type are intended as templates for your own specialized operations. Therefore, it's a good idea to change the generic Publish operation name to something more meaningful, such as PublishPurchaseOrder. It's also a good idea to remove operations that you don't expect process designers to use. For example, if you do not expect to use unparsed messages, you might want to remove them from your adapter type.

- Review template operations provided in Adapter for MQSeries Publish/ Subscribe and add new operations as necessary.

  The template operations provided with Adapter for MQSeries Publish/ Subscribe represent only a portion of the full range of operations you can implement using the Core class public methods combined with MQRFH class methods. For example, although template operations call the Core class getMessage and/or sendMessage functions only once (if at all) and create generic MQRFH instances, you can include operations that call these functions multiple times within an operation. This allows you to encapsulate complex messaging interactions within a single operation, such as one PAM business object to multiple MQSeries messages.

  For example, a PAM process might include a purchase order business object that corresponds to several MQSeries messages—one for the header, plus additional messages for each individual order line. The easiest way to implement this is to add a new basic operation (for example, GetPurchaseOrder) that calls getMessage multiple times in unparsed mode and copies information from the resulting variants into your Purchase Order business object.

TIP: Before you begin writing custom Java code for your adapter to deal with messages to/from publish/subscribe broker, you need to have a solid understanding of the message topics that applications are exchanging, which applications require which messages, and how the broker(s) are routing publications to their destinations. Information you need to have before coding includes the publication streams and subscription queues of your messages, their topics and the structure of their contents, and the queues on which broker replies appear.

For more information about using or implementing the MQSeriesAdapterCore and MQSeriesPubSubCustomExample classes, see the accompanying Javadoc reference. You might also want to review the MQSeriesPubSubCustomExample subclass, which illustrates how to implement the various custom methods contained in the MQSeriesAdapterCustom class.

Testing Adapter for MQSeries Publish/Subscribe

As a minimum, an adapter instance must meet these criteria before you can use it in an extension action. You must be able to:

- start up and shut down the adapter instance without error.
- verify that you can connect to the queue manager.

Before you release an adapter for general use by process designers, it's a good idea to test any custom code you have implemented. The easiest way to test custom code is to create a sample Partner Agreement Manager private process that uses extension actions that execute the adapter operations.

To test Adapter for MQSeries Publish/Subscribe:

1 Create a PAM private process that includes attempts to post and get a series of business objects. Make sure that you execute every path in your custom code.

2 After getting a business object, make sure—in VBScript (NT) or JavaScript (NT and UNIX)—that the contents are as you expect.

Tip: Use the StatusBO operation output and check the results.

3 Review the log file to make sure that no exceptions were reported within the adapter itself.

4 Turn on MQSeries tracing and look at the MQSeriesTrace.log file (located in the partner root directory) in the event of any MQSeries exceptions.

5 Test the public process events.

Adapter for MQSeries Publish/Subscribe reference

This section describes in more detail the sample Adapter for MQSeries Publish/Subscribe type that comes packaged with the product. The class names, properties, operations, and events are all described here.

Note: The operations are intended as templates only. You are encouraged to rename these operations and their parameters as needed, or design your own operations from scratch that use the MQSeriesAdapterCore class's methods in other ways.

Finally, since this sample adapter is really a subclass of the underlying sample Adapter for MQSeries Messaging type, its operations are also included.

NOTE: Each operation has a status BO that returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.

ADAPTER FOR MQSERIES PUBLISH/SUBSCRIBE TYPE

These are the components of the Adapter for MQSeries Publish/Subscribe type.

Class name	com.extricity.adapters.ibm.mqseries.pubsub.MQSeriesPubSub Adapter.
Description	This class represents the adapter implementation. It was generated from the Adapter Designer and subsequently edited to call methods in the MQSeriesAdapterCore class.
Class name	com.extricity.adapters.ibm.mqseries.pubsub.MQSeriesPubSubRFH.
Description	Class for managing MQ Rules and Format Headers of messages sent to or received from the broker. This class hides the complexity of managing the MQRFH fields and commands that are specific to the publish/subscribe MQRFH.
Class name	com.extricity.adapters.ibm.mqseries.pubsub.MQSeriesPubSub CustomExample.
Description	This class contains stub methods that can be implemented on site. These methods are as follows: ■ sendReply. Send a low-level acknowledgment to an incoming message. ■ receiveReply. Await a low-level acknowledgment to an outgoing message, including replies sent from the broker. ■ callInMap. Custom data transformation method for incoming messages; called whenever a “mapped” operation is indicated and the IncomingMap property is left null. ■ callOutMap. Custom data transformation method for outgoing messages; called whenever a “mapped” operation is indicated and the OutgoingMap property is left null. The default behavior of the mapping methods is to convert business objects to/from XML as well as read/write default MQRFHs.

Properties	QueueManager	Name of queue manager to connect to. **Mandatory**.
	Channel	Name of server connection channel by which to connect to queue manager. **Mandatory**.
	DefaultQueue	Name of a default queue to use for sending and receiving messages. Can be overridden in code. **Optional**.

Hostname	Name of host on which QueueManager resides. This can be an IP address or the IP host name. **Mandatory**.
Port	Name of port to which channel is listening. Defaults to 1424. **Mandatory**.
CorrelationID	The value is used only during event checking (that is, checkForEvents), otherwise, it is ignored. **Optional**.
MapWorking Directory	The location in the local file system to use for writing and reading files during a command-line mapping execution. Ignored if IncomingMap and OutgoingMap are blank. **Optional**.
IncomingMap	Command line that must be executed in order to map incoming messages. **Optional**. If left blank, the adapter calls the callInMap custom method instead for all incoming mapped messages. If populated, then mapped messages undergo the following steps: 1. Write incoming MQSeries message contents to a file, <MapWorkingDirectory>\MQmsg.txt (Windows NT) 2. Call the property value as a system command, which translates the MQSeries file to an XML file, <MapWorkingDirectory>\MQmsg.xml (Windows NT) 3. Read in the XML file and fill a business object with the contents. **Note:** If this mapping feature is used for publish/subscribe adapters, the map executable must be MQRFH-aware.
UserID	The user name used to connect to the queue manager. **Optional**, but must be entered if security is enabled on the server connection channel.
Password	The password associated with the user name. **Optional**, but must be entered if security is enabled on the server connection channel.

OutgoingMap	Command line that must be executed in order to map outgoing messages. **Optional.** If left blank, the adapter calls the callOutMap custom method instead for all outgoing mapped messages. If populated, mapped messages undergo the following steps: 1. Write BO to an XML file, <MapWorkingDirectory>\MQmsg.xml (Windows NT) 2. Call the property value as a system command, which translates the XML file to an MQSeries message file, <MapWorkingDirectory>\MQmsg.txt (Windows NT) 3. Read in the MQ file and put contents into an MQSeries message. **Note:** If this mapping feature is used for publish/subscribe adapters, the map executable must be MQRFH-aware.
MQSeriesTraceLevel	Sets the MQSeries trace level. One of six values: Off, or one of the five levels of detail that the MQSeries trace facility records debug-level information to the file MQSeriesTrace.log, located in the partner root directory. **Mandatory**.
WaitInterval	The length of time, in milliseconds, that an attempt to get a message from a queue must wait before erroring out, if a message is not immediately available. –1 indicates waiting indefinitely. **Mandatory**. Can be overridden in code.
BrokerControl Queue	The name of the queue to which to send broker control messages, such as publisher and subscriber registration/deregistration. **Mandatory**. Can be overridden in code.
PublisherStream	The name of the queue to which published messages must be sent. **Optional**. Can be overridden in adapter implementation.
SubscriberQueue	The name of the queue to which subscribed messages are to be sent. **Optional**. Can be overridden by adapter implementation.

	PublisherQueue	The queue to which a subscriber of a message must send a response to that message. **Optional**. The publisher awaits responses on this queue. The PublisherQueue also becomes part of the publisher's identity; subscribers who wish to receive messages from a publisher with PublisherQueue specified need to specify this queue as part of the subscription. Set this property if you expect subscribing applications to respond to messages that the adapter publishes.
	SubscriberStream	Limits the messages a subscriber receives to only those messages that a publisher sent to this specific publication stream. **Optional**. Only incoming publications sent to this stream are forwarded to the adapter.
	RegisterPublisherPer Message	Indicates whether or not to perform dynamic publication registration whenever a message is published, as opposed to registering the publisher ahead of time, upon startup. **Mandatory**. Can be overridden in the adapter implementation.
	CustomClass	Name of example Custom class. Must be the fully qualified class pathname, dot-delimited, relative to the classpath. Defaults to com.extricity.adapters.ibm.mqseries. MQSeriesPubSubCustomExample. **Mandatory**.
Operations	**GetUnparsed**	Retrieves a message and returns raw contents to the private process. Message is consumed in the queue.
	Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
	Output	Variant, representing message contents. Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
	Publish	Publish a message, with MQRFH.
	Input	The BO to publish. MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.

Output	Status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
BroadcastReply	Publish a message, with MQRFH, and await a single reply.
Input	The BO to publish. MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	The BO that contains the reply. Also, status BO.
BrowseUnparsed	Retrieves a message and returns raw contents to the private process. Message is not consumed in the queue.
Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Variant, representing message contents. Also, status BO.
GetMapped	Retrieves a message, calls a map, and returns the business object. Consumes message in the queue. NOTE: If the IncomingMap property is specified, the operation writes the incoming message to a file, calls the command specified in this property, reads in the resulting XML, and returns the BO. If this property is left null, calls the Custom callInMap method.
Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Business object. Also, status BO.
BrowseMapped	Retrieves a message, calls a map, and returns the business object. Does not consume message in the queue. NOTE: If the IncomingMap property is specified, the operation writes the incoming message to a file, calls the command specified in this property, reads in the resulting XML, and returns the BO. If this property is left null, calls the Custom callInMap method.

Input	MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Business object. Also, status BO.
PostUnparsed	Send a message, given the raw message contents.
Input	Variant, representing message contents MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Status BO.
PostMapped	Send a message using map to do data transformation. **NOTE:** If the OutgoingMap property is specified, the operation writes the BO as an XML file, calls the command specified in this property, reads the resulting bytes into a new MQMessage, and sends it. If this property is left null, calls the Custom callOutMap method.
Input	Business object. MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Status BO.
RequestUnparsed	Send a message (unparsed), then wait for a return message (unparsed).
Input	Variant, representing contents of outgoing message. MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.
Output	Variant, representing contents of reply message. Also, status BO.
RequestMapped	Send a message (mapped), then wait for a return message (mapped).
Input	Business object corresponding to output message. Map is applied to this BO as in PostMapped. MQSeriesMsgOptions business object containing values for the QueueName and CorrelationID to be used by the operation.

	Output	Business object corresponding to reply message. Map is applied to this BO as in GetMapped. Also, status BO.
Events	MQEvent. Contains Variant data. Functionality of checkForEvents is to poll a given queue and return any messages that appear, as in GetUnparsed.	

NOTE: Each operation has a status BO that returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.

CHAPTER

7 Using Adapter for MQSeries Workflow

Read this chapter to learn how to configure and use WebSphere® Partner Agreement Manager Adapter for MQSeries® Workflow version 2.1.

This chapter includes the following sections:

Adapter for MQSeries Workflow combines the process automation and tracking features of IBM MQSeries Workflow with the business-to-business integration capabilities of WebSphere Partner Agreement Manager.

NOTE: Partner Agreement Connect (PAC) is a version of Partner Agreement Manager (PAM) designed for partners who do not initiate processes. Because the same core documentation set is used for both PAM and PAC, the documentation uses the term "Partner Agreement Manager" or "PAM" to refer to both products.

About IBM MQSeries Workflow

IBM MQSeries Workflow provides process-oriented solutions on top of MQSeries. Processes consist of several activities that data passes through. This data is hierarchically structured, and closely matches the structure of PAM business objects. The activities represent programs running on client computers. These programs can link with end systems or present interfaces for user interaction (this is more common).

IBM MQSeries Workflow presents a client API in multiple languages—including Java—that you can use to implement a client application that both interacts with running processes and administers—starts or stops—them. Adapter for MQSeries Workflow is an example of such an application.

About Adapter for MQSeries Workflow

This version of Adapter for MQSeries Workflow is compatible with WebSphere Partner Agreement Manager (PAM) version 2.1. It requires the IBM MQSeries Workflow Java API, which is shipped with IBM MQSeries Workflow 3.3.

NOTE: Earlier versions of IBM MQSeries Workflow are not supported by this adapter.

Adapter for MQSeries Workflow consists of:

- class files corresponding to all Java classes.
- Java files for MQSeriesWorkflowAdapter and MQSeriesWorkflowAdapterCustom classes.
- Java and class files for an example MQSeriesWorkflowAdapterCustom subclass (MQSeriesWorkflowCustomExample.java and MQSeriesWorkflowCustomExample.class).

- XML representations of template adapter type, implementation, and instance (MQSeriesWorkflowAdapterType.xml, MQSeriesWorkflowAdapterJavaImp.xml, MQSeriesWorkflowAdapterInstance.xml).
- Javadoc reference for all class files.

Key features of Adapter for MQSeries Workflow include:

- *Ability to run an IBM MQSeries Workflow (MQWF) process from an extension action.* Certain operations create and start a process instance. These operations can also convert a PAM business object into an MQWF data container and pass it as the input to the new process. This feature allows an MQWF process to act as a subprocess in a PAM private process. You can also return immediately to PAM after submitting the MQWF process, waiting for it to finish.
- *Ability to embed PAM processes in an MQWF work item.* The adapter's checkForEvents method polls for work items belonging to a specific user. When a work item is found, its input container can be converted into a PAM business object and passed to PAM as an Adapter Server event. The work item is then programmatically closed out, allowing the MQWF process to continue. This feature allows a PAM public process to act as a subprocess in an MQWF process.
- *Ability to check status of MQWF processes.* This allows PAM users to design processes to react to changes in the state of MQWF processes—for example, to notify an administrator if an MQWF process has failed, or to go down a different path if an MQWF process hasn't completed yet.
- *Two kinds of data translation* are supported:
  - *Direct* isomorphic translation between PAM business objects and MQWF data containers that are identically named and structured.
  - *Custom* translation that allows you to implement your own transformation logic, which can include the direct translation of sub-elements.

- *Isomorphic* translation is the translation between a PAM business object and an MQWF data container in which the structures and element names are identical.

- *Per-adapter-instance mapping and network configuration.* Information that the adapter needs for successful operation, such as which maps to apply and how to connect to the MQWF execution server (domain name, system group name, system name), are set up as properties in the adapter instance. You can reuse the same adapter implementation, regardless of these property values.
- *All operations return a specialized Exception subclass.* This Exception differentiates between MQWF-related errors and non-MQSeries errors, and contains message and stack trace functionality that allows for better error reporting in Adapter Server dialog boxes and the PAM Auditor. In adapter operations, it's a good idea to catch this exception in your adapter implementations and populate a status business object.
- *Security* in the form of a user name and password that are entered as adapter properties.

About the Adapter for MQSeries Workflow environment

See *Installing Adapters for MQSeries* on page 5 for general system requirements. In addition to general requirements, you need:

- IBM MQSeries Workflow 3.3, which includes IBM MQSeries Workflow Java API and RMI-IIOP.
- IBM JDK 1.2.2 on the Adapter Server computer, and on the IBM MQSeries Workflow Java agent computer.
- IBM MQSeries Workflow Java agent up and running.

  It must be installed and running on the MQWF server computer and use the RMI-IIOP transport and locator options. The adapter can connect only if the gateway is up and running.
- IBM MQSeries Workflow JNDI name server, which must be set to connect to the IBM MQSeries Workflow Java agent. See IBM documentation to update the JNDI locator policy.

Updating the CLASSPATH

Adapter for MQSeries Workflow requires that certain IBM MQSeries Workflow Java agent jar files be present on the computer where the Adapter Server is running. The IBM MQSeries Workflow Java agent is included with IBM MQSeries Workflow 3.3, which is separately available from IBM.

After you install each of these products, you must move the required files to the computer where the Adapter Server is running. We recommend that you create a subdirectory under your partner root directory for third-party library files—for example, <partner_root>\external.

Product	Required jar file(s)	Installed in (default)
IBM MQSeries Workflow Java agent	fmcojagt.jar, fmcojapi.jar	C:\fmcwinnt\bin\java3300 (Windows NT)

To update your CLASSPATH on Windows NT:

1. From the Windows Start menu, choose Settings > Control Panel > System.
2. Click the Environment tab.
3. Select the CLASSPATH system variable.

   The current CLASSPATH value appears.
4. Edit this field as necessary to include the jar file.

   For example, if you copied the jar files to <partner_root>\external, the following must be in your CLASSPATH:
   <partner_root>\external\fmcojagt.jar;
   <partner_root>\external\fmcojapi.jar
5. Click Set, and then click Apply.

   The next time you launch the Adapter Server, it will recognize the MQSeries Java Client classes.

About configuring IBM MQSeries Workflow

Adapter for MQSeries Workflow is designed to require minimal setup in MQSeries.

- If you plan to use Direct translation, you must define MQWF data structures and PAM business objects in advance. Business objects and MQWF data structures must be isomorphic (identical in name and structure). For example, if a “submit process” adapter operation uses Direct translation, the input of the MQWF process must be a container that is identical to the operation’s input business object.
- If you plan to use checkForEvents, you must define MQWF program activities so that their input data structure is isomorphic to the PAM business object that corresponds to the Adapter Server event. Also, the name of the program activity (as it appears in the MQWF process, *not* the name of the program itself) must match the name of the Adapter Server event.
- You might want to create an MQWF user exclusively for Partner Agreement Manager adapters. When Adapter for MQSeries Workflow connects to an MQWF domain, it uses a specific user name, and its checkForEvents checks for any work items with a specific name visible to that user. Therefore, you must make sure that work items do not get assigned to that user if they are not intended for the adapter.
- The MQWF program activity that corresponds to Adapter Server events must have these properties:
  - It must be set for manual invocation, so that it is in a wait state until the adapter’s checkForEvents can pick up its input. If it is invoked automatically, the adapter might never “see” it.

  - There is no need to implement an executable for this program. (Use a harmless executable, such as java.exe without any arguments.) The adapter's checkForEvents force-finishes the work item once it has read the input container; consequently the specified executable never executes.

- If you use an MQWF process as a PAM subprocess that is set so that the extension action waits for the MQWF process to complete, you might need to return the output of the MQWF process to PAM. The only way to do so is via a work item that the extension action looks for (much like checkForEvents looks for work items). Make sure that this work item is identified with a name that differs from work items that checkForEvents picks up. Also, this should be the last activity in the process, so that its output really represents the finished state of the MQWF process.

Installing Adapter for MQSeries Workflow

You must install Adapter for MQSeries Workflow—on the same computer where the Adapter Server is running—by copying the appropriate java and class files into your PAM partner directory.

Warning: Make sure you *back up* your partner root directory *before* installing the adapter. This is especially important if you have ever used (and customized) previous versions of MQSeries adapters.

Before you install Adapter for MQSeries Workflow, make sure that:

- the IBM MQSeries Workflow execution server is installed and configured correctly. Although you can run both the IBM MQSeries Workflow execution server and the Adapter Server on the same computer, we strongly recommend that you install them on different computers. The two computers must be able to communicate via TCP/IP.
- the Java gateway is configured for RMI-IIOP communication.
- the IBM MQSeries Workflow Java API is installed on the computer where the Adapter Server is running, and that the Adapter Server's CLASSPATH is set to include the Java API class files (see *Installing Adapters for MQSeries* on page 5).

To install Adapter for MQSeries Workflow:

1. Log on—as a user with administrative privileges—to the computer where the Adapter Server is installed.
2. Add the following libraries to the client's CLASSPATH:
   - MQSeries support library (com.ibm.mq.jar)
   - Workflow library (fmcojagt.jar, fmcojapi.jar)
3. Install Adapter for MQSeries Workflow.

   On Windows NT:

   A. Copy MQSeriesWorkflow2_1.zip to your partner root directory.

   B. Extract the zip file into your partner root directory.

IMPORTANT: If you previously installed an MQSeries adapter, a Confirm File Overwrite dialog appears. Click Yes to All (twice) to replace the existing files.

NOTE: If you have the PAM client and server on two different computers, you need to extract the files into the partner directory on both computers.

Adapter for MQSeries Workflow files appear in this location:

On this platform:	The files are located in:
Windows NT	<partner_root>\com\extricity\adapters\ibm\mqwf

Make sure that the java and class files are present in the same directory.

Configuring Adapter for MQSeries Workflow

After you install Adapter for MQSeries Workflow, you can configure it and take advantage of its default behavior, or you can modify the adapter to suit your specific needs (see *Modifying Adapter for MQSeries Workflow* on page 111). For general importing guidelines, see *Importing key elements* on page 7.

To configure Adapter for MQSeries Workflow:

1 Implement any custom code by writing one or more MQSeriesWorkflowAdapterCustom classes or subclasses.

TIP: If you plan to use the default mapping behavior of converting business objects directly to/from MQWF data container objects, do not edit the MQSeriesWorkflowAdapterCustom class. See *Modifying Adapter for MQSeries Workflow* on page 111 for tips on implementing custom code.

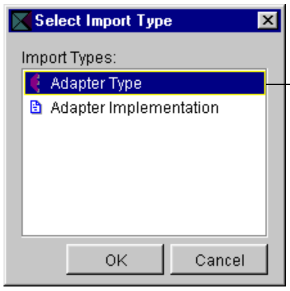
2 If the Adapter Server is not already running, start it.

3 Start the Partner Agreement Manager Process Server and the Adapter Server Administrator.

4 Choose Adapter Designer from the Tools menu to open the Adapter Designer.

5 In the Adapter Designer, choose Import from the File menu.

The Select Import Type dialog box appears.



You can import either an adapter type or an adapter implementation.

6 Select Adapter Type and click OK. In the Import dialog box, select the MQSeriesWorkflow Adapter Type XML file located in:
`<partner_root>\com\extricity\adapters\ibm\mqwf\test`

This creates a template adapter type that you can edit to conform to your customized adapter code. For example, you can add any new operations you need.

- Edit the inputs and outputs or operations to match the business objects you plan to get and post.
- Follow the instructions in the *Partner Agreement Manager Adapter Developer's Guide* to set default property values as appropriate for the adapter type.

- If you plan to use checkForEvents, you must create an appropriate event type. Its name must match the name of the corresponding MQWF program activity, and its business object must map (either Direct or in your Custom code) from the input to this activity.

7. In the Adapter Designer, choose Import again, select Adapter Implementation, and select the MQSeriesWorkflow Adapter Java Imp XML file located in: `<partner_root>\com\extricity\adapters\ibm\mqwf\test`

   This creates the adapter implementation. Make sure the adapter implementation's class name and package names are correct.

8. In the Adapter Server Administrator, choose Adapter Manager from the Tools menu to start the Adapter Manager.

9. Choose Add from the Adapter menu to create a new instance of Adapter for MQSeries Workflow, and follow the instructions in the *Partner Agreement Manager Adapter Developer's Guide* to set the connection property values.

   - If your adapter checks for events, make sure that event polling is enabled and polls at an appropriate time interval.
   - If you have implemented custom mapping methods, set the TranslationStyle property to "Custom."
   - If you have created a subclass of the MQSeriesWorkflowAdapterCustom class for this instance, you must enter its name in the CustomClass property.
   - Set the DebugMessaging and MQSeriesTrace properties according to your debugging needs. You might want to turn these on only during development, testing, or maintenance. The logs associated with these properties can quickly grow very large.
   - Click the Environment tab and make sure the Use System's Classloader setting is turned on.

10. In the Adapter Manager, start the new adapter instance and resolve any exceptions that occur.

    As soon as the adapter instance has connected to the queue manager, you are ready to begin using it in extension actions.

Modifying Adapter for MQSeries Workflow

Although you can use Adapter for MQSeries Workflow as is, the default behavior and settings might not meet your business needs. For example, the default mapping behavior for mapped operations is to use Direct translation.

As a result, you might want to customize Adapter for MQSeries Workflow to suit your specific needs.

Important: See the *Partner Agreement Manager Adapter Developer's Guide* for more information about customizing adapters.

Before you begin writing custom Java code for your adapter, make sure that you understand:

- The layout of your MQWF network, including the domain -> system group -> system hierarchy and how Adapter for MQSeries Workflow communicates with it.
- The MQWF staff requirements and how they relate to PAM.

  Make sure that the user (user name and password) that the adapter uses sees only those work items that are intended for PAM.

  Make sure you understand how this requirement fits in with other MQWF staff requirements.
- The formats of MQWF data containers that PAM sends and receives, and how these messages are to be converted to and from business objects.
  - What are the message contents?
  - Can third-party mapping and data transformation tools be used for MQWF-to-XML mapping?
  - Which of the two basic data transformation types are more appropriate for each data structure?
  - Is Direct mapping sufficient? If not, can you still perform Direct mapping to an "intermediary" business object (or specific elements of the business objects), and then use a Map or Script step in the private process? Or is coding in a Custom class method more appropriate?
  - When determining how to do the data mapping, consider not only the data format and available mapping capabilities, but also whether it's appropriate for process designers to do the mapping in the private process.

- How MQWF processes interact with PAM processes.

  Which is a subprocess of the other? Or are they subprocesses at all? If there is a subprocess relationship, make sure that you understand which data is passed from one process to the other.

You should bear in mind the following tips during implementation:

- Customize the operations in the adapter type to make it easier for process designers to select operations correctly.

  The PAM process builders who will use this adapter in private process extension actions rely on the names of the operations and their inputs and outputs to identify the correct operation. The operations provided with Adapter for MQSeries Workflow type are intended as templates for your own specialized operations. Therefore, it's a good idea to change the generic operation names to something more meaningful. It's also a good idea to remove operations that you don't expect process designers to use.

- Review template operations provided in Adapter for MQSeries Workflow and add new operations as necessary.

  The template operations provided with Adapter for MQSeries Workflow represent only a portion of the full range of operations you can implement.

For more information about using or implementing the MQSeriesWorkflowAdapterCore and MQSeriesWorkflowCustomExample classes, see the accompanying Javadoc reference. You might also want to review the MQSeriesWorkflowCustomExample subclass, which illustrates how to implement the various custom methods contained in the MQSeriesWorkflowAdapterCustom class.

Testing Adapter for MQSeries Workflow

As a minimum, an adapter instance must meet these criteria before you can use it in an extension action. You must be able to:

- start up and shut down the adapter instance without error.
- verify that you can connect to the queue manager.

Before you release an adapter for general use by process designers, it's a good idea to test any custom code you have implemented. The easiest way to test custom code is to create a sample Partner Agreement Manager private process that uses extension actions that execute the adapter operations.

To test Adapter for MQSeries Workflow:

1. Create a PAM private process that includes attempts to post and get a series of business objects. Make sure that you execute every path in your custom code.
2. After getting a business object, make sure—in VBScript (NT) or JavaScript (NT and UNIX)—that the contents are as you expect.

   TIP: Use the StatusBO operation output and check the results.

3. Review the log file to make sure that no exceptions were reported within the adapter itself.
4. While debugging, try setting the adapter's DebugMessaging property to Terse or Verbose.
5. Turn on MQSeries tracing and look at the MQSeriesTrace.log file (located in the partner root directory) in the event of any MQSeries exceptions.
6. Test the public process events.

ADAPTER FOR MQSERIES WORKFLOW REFERENCE

This section describes in more detail the sample Adapter for MQSeries Workflow type that comes packaged with the product. The class names, properties, operations, and events are all described. Note that the operations are intended as templates only. You are encouraged to rename these operations and their parameters as needed, or design your own operations from scratch that use the MQSeriesWorkflowAdapterCore class's methods in other ways.

NOTE: Each operation has a status BO that returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.

ADAPTER FOR MQSERIES WORKFLOW TYPE

These are the components of Adapter for MQSeries Workflow.

Class name	com.extricity.adapters.ibm.mqwf.MQSeriesWorkflowAdapter
Description	This class represents the adapter implementation. It was generated from the Adapter Designer and subsequently edited to call methods in the MQSeriesWorkflowAdapterCore class.

Class name	com.extricity.adapters.ibm.mqwf.MQSeriesWorkflowAdapterCore
Description	This represents the bulk of the MQSeries Workflow logic. It includes the following features: ■ Connect to and disconnect from an MQWF execution server's Java gateway given connection information specified in the instance's properties or passed as parameters to given methods. ■ Dispatch the task of mapping between business objects and MQWF data container objects. ■ Process instance control, including starting, checking status, and waiting to finish. ■ Checking for work items, both process outputs and activities intended for checkForEvents. ■ Throw exceptions containing meaningful error text when error conditions are found. ■ Call methods in the Custom class.
Class name	com.extricity.adapters.ibm.mqwf.MQSeriesWorkflowAdapter Exception
Description	This exception subclass can be thrown by most methods, and contains information on error conditions. It also differentiates between those exceptions that are thrown by the MQWF Java API and those that are thrown by adapter code.
Class name	com.extricity.adapters.ibm.mqwf.MQSeriesWorkflowTransform
Description	Performs direct, isomorphic translation between PAM business objects and MQWF data container objects. These methods are implemented as statics and can be called from anywhere, including Custom map methods.
Class name	com.extricity.adapters.ibm.mqwf.MQSeriesAdapterCustom
Description	This class contains stub methods that can be implemented on site. These methods are as follows. (1) callInMap. Custom data transformation method for incoming data. (2) callOutMap. Custom data transformation method for outgoing data.

Properties	DomainName	Name of MQSeriesWorkflow domain to connect to. In form of iiop://<host_name>:<port>/<MQWF agent_name> where: host_name - of MQWF Java agent; port - that the Java agent is configured to use; agent_name - name of Java agent. **Mandatory**.
	SystemGroup	Name of the system group. **Mandatory**.

	System	Name of the system on which the MQWF execution server's Java gateway is running. **Mandatory**.
	UserID	Name of the MQSeriesWorkflow user. Must be capitalized. **Mandatory**.
	Password	Password associated with the UserID. Case-sensitive. **Mandatory**.
	TranslationStyle	Indicates whether Direct or Custom mapping is to be used for all operations that accept or return business objects. If Direct, then the Core class calls MQSeriesWorkflowTransform methods in order to translate between PAM business objects and MQWF data container objects. If Custom, then the Core class calls the Custom mapping methods. **Mandatory**.
	DebugMessaging	One of three values: Off, Terse, Verbose, representing different levels of debug messaging. Messages are written to the Adapter Server console. Debug messaging can be useful during development and deployment, but is generally set to Off in a production environment. **Mandatory**.
	CustomClass	Name of Custom class. Must be the fully qualified class pathname, dot-delimited, relative to the classpath. Defaults to com.extricity.adapters.ibm.MQSeries.MQSeries WorkflowAdapterCustom. **Mandatory**.
Operations	SubmitProcess	Submits a process and returns immediately after submission. An input BO is translated and passed as the input to the process.
	Input	Business object
	Output	Variant representing the unique name of the process instance. Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.
	SubmitProcessAnd Wait	An "Advanced" variation of the SubmitProcess operation, submits a process and waits for it to complete, translating and returning any output data.
	Input	Business object
	Output	Returned business object. Also, status BO, which returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.

	CheckProcessState	This simple operation returns the status (variant) of a given named process instance. Adapter developers and/or process designers can use this feature to implement conditional logic based on the status of a given process instance.
	Input	Variant representing the unique name of the process instance whose status must be checked.
	Output	Variant indicating status, one of: Deleted, Finished, Ready, Running, Suspended, Suspending, Terminated, Terminating, Undefined. Also, a status BO.
	RetrieveProcess Output	If a process was submitted earlier without waiting for it to complete, the receipt of any output data can be performed later via this “Get” operation.
	Input	Variant representing the unique name of the process instance whose status must be checked.
	Output	Returned business object. Also, a status BO.
	Ping	Tests that the connection with the MQWF server’s Java gateway is live. If the connection has been idle for several hours, the MQWF server might disconnect without notifying the client. This Ping operation performs some stateless interactions that ensure that the connection is live and that data can be passed across.
	Input	None
	Output	Status BO
Events	ExampleProgram. A sample event illustrating how checkForEvents checks for open work items and generates Adapter Server events out of them.	

NOTE: Each operation has a status BO that returns either result="success" or result="failure". If the operation fails, the status BO also returns the specific reason why it failed.

APPENDIX

A

NOTICES

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

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information 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.

Trademarks

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

AIX
DB2
IBM
MQSeries
SupportPac
WebSphere

Pentium is a registered trademark of Intel Corporation in the United States and/or other countries.

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

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

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

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

Glossary

action—a task performed as part of a private process. A private process action is the equivalent of a step in a public process. See the following terms in this glossary for more information about the action types you can include in a private process:

- approval action
- extension action
- mapping action
- notification action
- output object action
- script action
- subprocess action
- termination action
- timer action

See also *private process*.

adapter—the software bridge between Partner Agreement Manager processes and specific end-system and business-application interfaces. Adapters manage interactions between business applications and the Adapter Server. They allow private processes to interact with external business applications while a process is running, and they allow PAM to start public processes based on events that occur in external business applications. See also *adapter implementation*, *adapter instance*, *adapter type*.

adapter implementation—the implementation declaration for an adapter type. It specifies the name and location of the Java source file that defines the application logic used to communicate with a specific end system through that end system's interface. The application logic is specified in the form of properties. See also *adapter*, *adapter instance*, *adapter type*.

adapter instance—an instance of an adapter implementation. The adapter instance is used in a private process extension action and provides the specific values to be used for the properties declared in the adapter implementation. See also *adapter*, *adapter implementation*, *adapter type, extension action*.

adapter type—a definition that is stored in XML format and specifies the adapter's properties as well as the operations and events it supports. A single adapter type can have multiple implementations, and each implementation can have multiple instances. See also *adapter*, *adapter implementation*, *adapter instance*.

approval action—a private process action that you use to ask for a response from a user before letting the process continue to run. You can use an approval action, for example, to ask for an OK when a purchase order exceeds a predetermined amount. See also *private process*.

business object—a message transmitted as part of a public process. Business objects take the form of purchase orders, acknowledgments, requests for clarification, and so on. See also *business object type*.

business object type—a definition that determines the types of information a message can contain. It has three properties: the top-level element in its element definition set, its key field, and whether instances of it return audit information for non-repudiation purposes. The name of the business object type is the name of the element you select as its top-level element. See also *business object*, *element definition set*, *non-repudiation*.

business object variable—one of the two types of variables used in Partner Agreement Manager to store information within a process. Business object variables create an instance of a business object type. They can be used to store, for example, the outputs from extension actions, the inputs for map actions, or the inputs and outputs for subprocesses. See also *business object*, *business object type*, *extension action*, *variant variable.*

CA—see *certificate authority*.

certificate—a security document that binds a public encryption key to an entity (an individual or organization) known as the principal. The security document (a digital certificate) is signed by another entity known as the issuer. A digital certificate for which both the principal and issuer are the same entity is known as a self-signed certificate. A certificate for which the principal and issuer are different entities is issued by a certificate authority (CA) like VeriSign and is known as a CA-issued (or third-party-signed) certificate. Partner Agreement Manager supports both self-signed and CA-issued certificates. PAM also supports the binding of certificates to be used for signature authentication, message encryption, and SSL authentication for channels other than Partner Agreement Manager. See also *certificate authority*, *SSL*.

certificate authority—a trusted third-party organization or company that issues digital certificates used to create digital signatures and public-private key pairs. The role of the certificate authority, or CA, is to authenticate the entities (individuals or organizations) involved in electronic transactions. CAs are a critical component in data security and electronic commerce because they guarantee that the two parties exchanging information are really who they claim to be. See also *certificate*.

channel—a communications mechanism that encapsulates all the processing information needed to send messages to a partner's system, as well as to translate data received from a partner into Partner Agreement Manager messages. PAM provides channels for RosettaNet, EDI, cXML, and other systems and protocols. See also *message*.

digital certificate—see *certificate*.

DTD—Document Type Definition. A type of file associated with SGML and XML documents that defines how the formatting tags should be interpreted by the application presenting the document. In Partner Agreement Manager, a DTD file contains the complete description of a business object type's element definition set. See also *business object*, *business object type*, *element definition set*.

element definition set—a collection of data fields (or elements) or groups of data fields that defines the structure and meaning of a business object type. See also *business object*, *business object type*.

encryption certificate—see *certificate*.

event—a piece of information that comes into Partner Agreement Manager as a message from another source (an enterprise system or business application, for example) and triggers a public process. See also *message*.

event push—a method that uses the HTTP POST mechanism to push events into Partner Agreement Manager as a way to trigger processes. A port on the Process Server is set to listen for events in the form of HTTP POST messages. When a message is detected, PAM uses the information in the message to generate an event. See also *event*.

extended enterprise—a business model under which companies that work together as partners function as efficiently as a single organization through the implementation of automated communication technologies.

extension action—a private process action that communicates via an adapter with an external application that is registered with Partner Agreement Manager. You can use an extension action, for example, to launch a spreadsheet application, perform calculations, and update the enterprise system, or to get information from an enterprise system or listen for an event in the enterprise system. See also *adapter*, *private process*.

LDAP—Lightweight Directory Access Protocol. LDAP provides a standard method for accessing information from a central directory. After user authentication is set up in the LDAP directory, applications that use the LDAP protocol can retrieve the information from that directory. An authenticated user can log in to any application that supports the LDAP protocol with the same user name and password.

linked certificate—see *certificate*.

map—a Java Script or VBScript that inserts data into fields in an output business object type generated by a private process. The map specifies which fields in the output business object type receive data, and it identifies the information source.

map method—a reusable logical block of code that inserts data into a particular type of element or element sequence in a business object type. Within a map method, you can write the expressions that map individual input and output fields in the sequence. Or you can create a submap and drag input fields to output fields and have Partner Agreement Manager create the appropriate mapping expressions. See also *map*, *submap*.

mapping action—a private process action that you use to call a map. The map specifies the fields in a business object type that will receive data extracted from another source. You use a mapping action when you want to extract data from one business object type and insert it in a different business object type. For example, you use a mapping action to transform a purchase order generated by your inventory system into a sales order in a format that your partner expects. See also *map*, *private process*.

message—a structured communication used to pass information and control to another partner in a public process. The action in the process passes to the partner who receives the message. The content of a message is determined by its business object type. A message can be transmitted via synchronous or asynchronous methods, as determined by its communication service type. See *business object type*.

non-repudiation—a business object security feature that authenticates instances of a business object type and maintains an audit record to verify that they were received by the intended recipient. For business object instances that you receive, Partner Agreement Manager authenticates each instance and maintains an audit record to verify that the instance actually originated with the stated originator. If you disable auditing for a business object type, non-repudiation support is disabled for all messages that contain instances of that business object type.

notification action—a private process action that you use to send an e-mail, fax, or pager message to addressees that you specify. You use a notification action to inform someone inside or outside your organization that an event has occurred. For example, you can use a notification action to alert the order entry department when a purchase order arrives from a customer. See also *private process*.

output object action—a private process action that you use to bind a business object to the expected output object and path in a public process. You use an output object action at the point in a private process when you are ready to send a business object to the associated public process. This is typically the last action in the private process. See also *private process*.

partner group—a group of partners that perform the same role in a process at different times. Instead of duplicating a public process and substituting a different partner name, you can set up a partner group for the public process and then designate a specific partner as the participant when you start an instance of the process. For example, you might design a generic purchasing process that works equally well with any of your suppliers and then designate the appropriate partner when you start the process.

partner profile—information that identifies an organization, specifies a contact person in that organization, lists the communication services the organization supports, and defines the organization's security profile. When partners agree to participate in a public process, they must exchange profile information as a way to ensure authenticity before they can proceed.

PIP—Partner Interface Process. RosettaNet PIPs are specialized system-to-system XML-based dialogs that define business processes between supply-chain partners and provide models and documents for the implementation of e-commerce standards. Each PIP includes a technical specification based on the RosettaNet Implementation Framework (RNIF), a message guideline document with a PIP-specific version of the business dictionary, and an XML message guideline document. See also *RosettaNet*.

post method—the last block of code that is executed when a mapping action runs. Its only parameter is the output business object. You use the post method when you need to perform post-processing on the output business object. For example, you might use the post method to set the value of a summary field based on the number of line items in the output business object, or to examine a range of dates in a repeated group, extract the most recent date, and post that date in a header field. See also *mapping action*, *pre method*.

pre method—the first block of code that is executed when a mapping action runs. The pre method's parameters are the map inputs. You use the pre method to access a map's inputs and set global variables based on their content. See also *mapping action*, *post method*.

private process—a task or set of tasks that business partners participating in a public process perform at points where they need to take action internally. Partners participating in a public process must implement a private process for each public process step that they own. A private process begins with input from the public process and ends with output that feeds back into the public process. The input can be the receipt of a business object from a partner, or it can be a triggering event from an internal system. The output is the business object that transfers control back to the public process. See also *action*, *process*, *public process*.

private process action—see *action*.

process—the flow of actions and the exchange of business information between partners in an extended enterprise. A process operates on two levels, public and private. See *extended enterprise*, *private process*, *public process*.

public process—the step-by-step flow of messages, events, and actions between two or more business partners. Public processes are set up by agreement between partners, and each step in a public process has a private process associated with it. A public process is developed by one partner, and all the partners who participate in it must review and approve it before it can be implemented. The partner who designs a public process is its owner. See also *private process*, *process*.

RosettaNet—a consortium of major information technology, electronic components, and semiconductor manufacturing companies that is working to create and implement industry-wide, open e-business process standards. See also *PIP*.

script action—a private process action that consists of a script written in VBScript or JavaScript and is designed to manipulate information or set up conditional actions based on input. You use a script to establish decision-making criteria for branches or loops, to set variables, or to calculate values that are used elsewhere in the private process. See also *private process*.

security certificate—see *certificate*.

self-signed certificate—see *certificate*.

signature certificate—see *certificate*.

SSL—Secure Sockets Layer. The SSL protocol is a security protocol that provides for communications privacy and reliability over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

submap—a secondary level map that is called by a map method to insert data into an output element other than the top-level element. See *map*, *map method*.

subprocess action—a private process action you use to call an existing public process. You can call any public process in which your organization owns the first partner action. For example, you can use a subprocess to get a quote approved by a third-party supplier before responding to a customer. See also *private process*.

termination action—a private process action that you use to stop a process at a predetermined point for a reason that you specify. You can use a termination action to deal with errors in data that might prevent a process from completing successfully. For example, you might want to stop a process in cases where an enterprise system passes incomplete or corrupted information to it. See also *private process*.

third-party-signed certificate—another name for a CA-issued certificate. See *certificate*.

timer action—a private process action that you use to insert a pause. You can use a timer action to specify the period of time you want to elapse before the next action in the process starts. See also *private process*.

variant variable—single field variables. Variant variables store text strings—the type of information contained in a single field element. You can use variant variables to store the input for actions, to set flags (such as the time-out flag for an approval action), to move information within scripts, or to store the results of an approval action. See also *business object variable*.

INDEX