BEA WebLogic Adapter for MQSeries®

User Guide

Version 8.1.0
Document Revised: July 2003

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

About This Document

1. Introducing the BEA WebLogic Adapter for MQSeries

2. Defining Application Views for MQSeries

3. Using the Adapter for MQSeries

About This Document

This document describes how to install, configure, and use the BEA WebLogic Adapter for MQSeries. This document is organized as follows:

- Chapter 1, "Introducing the BEA WebLogic Adapter for MQSeries," describes the adapter, and how it relates to both MQSeries and WebLogic Integration.
- Chapter 2, "Defining Application Views for MQSeries," describes application views and how to use them to configure events and services.
- Chapter 3, "Using the Adapter for MQSeries," describes how to use the services and events of the adapter in a business process.
- Appendix A, "Request and Response Schemas," presents sample request and response schemas for the services and the events.
- Appendix B, "Run-Time Parameter Values," presents the values required for configuring each service during run-time.
- Appendix C, "Error Messages and Troubleshooting," provides details on errors messages and troubleshooting.

Who Should Read This Documentation

This document is intended for the following members of an integration team:

- Integration Specialists—Lead the integration design effort. Integration specialists have expertise in defining the business and technical requirements of integration projects, and in designing integration solutions that implement specific features of WebLogic Integration.

The skills of integration specialists include business and technical analysis, architecture design, project management, and WebLogic Integration product knowledge.

- Technical Analysts—Provide expertise in an organization's information technology infrastructure, including telecommunications, operating systems, applications, data repositories, future technologies, and IT organizations. The skills of technical analysts include technical analysis, application design, and information systems knowledge.
- Enterprise Information System (EIS) Specialists—Provide domain expertise in the systems that are being integrated using WebLogic adapters. The skills of EIS specialists include technical analysis and application integration design.
- System Administrators—Provide in-depth technical and operational knowledge about databases and applications deployed in an organization. The skills of system administrators include capacity and load analysis, performance analysis and tuning, deployment topologies, and support planning.

Additional Information

To learn more about the software components associated with the adapter, see the following documents:

- *BEA WebLogic Adapter for MQSeries Release Notes*

  http://edocs.bea.com/wladapters/mq/docs81/pdf/relnotes.pdf
- *BEA Application Explorer Installation and Configuration Guide*

  http://edocs.bea.com/wladapters/bae/docs81/index.html
- *Introduction to the BEA WebLogic Adapters*

  http://edocs.bea.com/wladapters/docs81/pdf/intro.pdf
- BEA WebLogic Adapters 8.1 Dev2Dev Product Documentation

  http://dev2dev.bea.com/products/wladapters/index.jsp
- Application Integration documentation

  http://edocs.bea.com/wli/docs81/aiover/index.html

  http://edocs.bea.com/wli/docs81/aiuser/index.html

- BEA WebLogic Integration documentation

  http://edocs.bea.com/wli/docs81/index.html

- BEA WebLogic Integration topics in the WebLogic Workshop help system

  http://edocs.bea.com/workshop/docs81/doc/en/core/index.html

- BEA WebLogic Platform documentation

  http://edocs.bea.com/platform/docs81/index.html

- MQSeries documentation

  http://www.ibm.com/

How to Use This Document

This document is designed to be used in conjunction with *Using the Application Integration Design Console*, available at the following URL:

http://edocs.bea.com/wli/docs81/aiuser/index.html

Using the Application Integration Design Console describes, in detail, the process of defining an application view, which is a key part of making an adapter available to process designers and other users. What *Using the Application Integration Design Console* does *not* cover is the specific information about Adapter for MQSeries that you need to supply to complete the application view definition. You will find that information in this document.

At each point in *Using the Application Integration Design Console* where you need to refer to this document, you will see a note that directs you to a section in your adapter user guide, with a link to the edocs page for adapters. The following roadmap illustration shows where you need to refer from *Using the Application Integration Design Console* to this document.

Figure 1 Information Interlock with *Using the Application Integration Design Console*

Contact Us!

Your feedback on the BEA WebLogic Adapter for MQSeries documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the BEA WebLogic Adapter for MQSeries documentation.

In your e-mail message, please indicate that you are using the documentation for BEA WebLogic Adapter for MQSeries and the version of the documentation.

If you have any questions about this version of BEA WebLogic Adapter for MQSeries, or if you have problems using the BEA WebLogic Adapter for MQSeries, contact BEA Customer Support through BEA WebSUPPORT at **www.bea.com**. You can also contact Customer Support by using the contact information provided on the Customer Support Card which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

Convention	Item

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

Convention	Item

. . .	Indicates one of the following in a command line:
	• That an argument can be repeated several times in a command line
	• That the statement omits additional optional arguments
	• That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

CHAPTER 1

Introducing the BEA WebLogic Adapter for MQSeries

This section introduces the BEA WebLogic Adapter for MQSeries and describes how the adapter enables integration with MQSeries business objects and WebLogic Integration.

It includes the following topics:

About Adapters and BEA WebLogic Integration

The BEA application integration solution uses adapters and application views to help you integrate applications in your enterprise. At its most fundamental level, an *adapter* is software that connects an enterprise information system (EIS) and an integration server. This bi-directional connection consists of *services*—interactions that originate in the adapter (and may require a response from the EIS)—and *events*, interactions that originate in the EIS.

Most EIS systems make selected information and functions available to other applications by way of specialized integration APIs. An adapter connects to the EIS through its integration API, or through database or system calls, and exposes the available functions from the EIS. However, rather than exposing the intricacies of APIs to users, WebLogic Integration incorporates *application views*—business-oriented interfaces that provide a layer of abstraction between an adapter and the EIS capabilities the adapter exposes.

Figure 1-1 Application View in an Integration Solution

Application views contain definitions for the services and events used by business processes to communicate with an EIS. They also contain connection information and XML schema that define inputs and outputs for services and events. After an adapter is deployed, you can use its Web-based interface to define as many applications views as you need, and other WebLogic Integration components and applications can use that adapter to access data on the EIS.

To learn more about BEA WebLogic Integration in the BEA WebLogic Workshop environment, see the WebLogic Integration site at the following URL:

http://edocs/wli/docs81/index.html

To learn more about adapters in general, see *Introduction to the BEA WebLogic Adapters for WebLogic Integration* at the following URL:

`http://edocs.bea.com/wladapters/docs81/pdf/intro.pdf`

To learn more about the role of adapters in application integration architecture, see "Key Components of Integration Solutions" on page 1-3.

Key Components of Integration Solutions

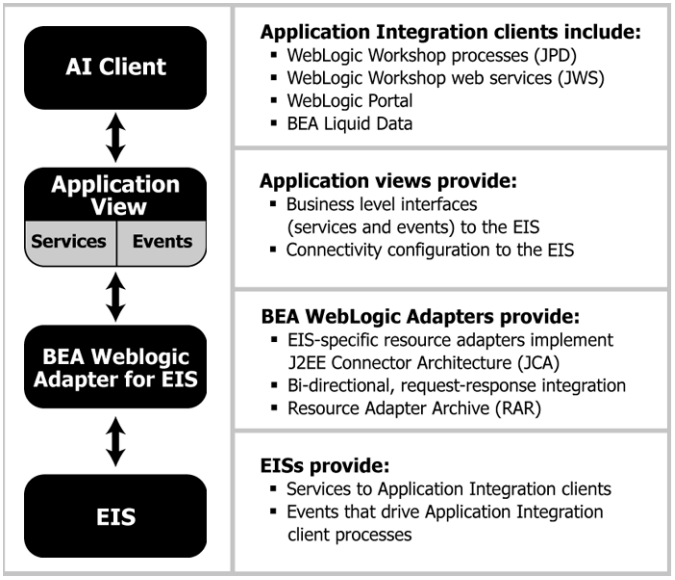
This section describes some of the key concepts you need to be familiar with before you work with an adapter.

- Basic WebLogic Integration Architecture
- Enterprise Information Systems
- Resource Adapters
- Application Views
- Service Clients and Event Consumers

Basic WebLogic Integration Architecture

Adapters are used in conjunction with the Application Integration component of BEA WebLogic Integration. This component provides a systematic, standards-based architecture for hosting business-oriented interfaces to enterprise applications.

Figure 1-2 Adapters in the Application Integration Architecture



For general information about Application Integration, see the following documents:

- *Introducing Application Integration* at the following URL:

  `http://edocs.bea.com/wli/docs81/aiover/index.html`

- *Using the Application Integration Design Console* at the following URL:

  `http://edocs.bea.com/wli/docs81/aiuser/index.html`

Enterprise Information Systems

An *enterprise information system* (EIS) is software that provides the information infrastructure for an enterprise. An EIS offers a set of services to its clients, which are made available to clients via local and/or remote interfaces. An integration solution involves integration with one or more EISs.

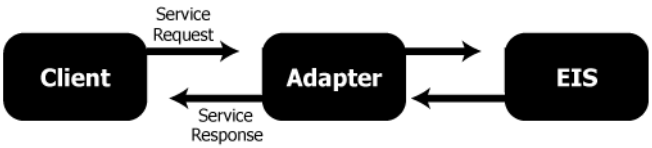
Resource Adapters

A *resource adapter* (or simply *adapter*) is a BEA software component that acts as a connector between an EIS and a J2EE application server (such as BEA WebLogic Server). Each adapter provides bi-directional, request-response integration with a specific application or technology.

Adapters handle two general types of operations:

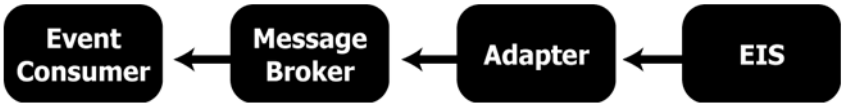
- *Services* are request/response communications with the EIS. Client applications submit service requests to the EIS via the adapter, and the adapter returns the EIS response back to the client. For example, a business process might place an item on an MQSeries queue or execute a SELECT statement on a database. Responses are either synchronous or asynchronous.

Figure 1-3 Service Invocations



- *Events* are asynchronous, one-way messages received from an EIS. For example, the adapter can receive a message from an MQSeries system. The adapter routes the EIS message to the appropriate software component via the WebLogic Integration Message Broker and the Application Integration JMS infrastructure.

Figure 1-4 Event Notifications



In effect, a service is a *request for some work to be done* and an event is a *notification that some work has been done*.

For more information about the specific services and events supported by the Adapter for MQSeries, see "About the BEA WebLogic Adapter for MQSeries" on page 1-7.

To learn more about the WebLogic Integration Message Broker and the Application Integration JMS infrastructure, see *Introducing Application Integration* at the following URL:

`http://edocs.bea.com/wli/docs81/aiover/index.html`

Application Views

An *application view* is a business oriented interface to objects and operations within an EIS. Application views include the information needed to communicate with the EIS as well as configurations for services and events.

To learn more about using application views in processes, see the "WebLogic Workshop Development Environment" at the following URL:

`http://edocs.bea.com/workshop/docs81/doc/en/core/index.html`

You typically define an application view for a specific business process. Therefore, you might have multiple application views defined for a single adapter, each designed to meet a specific requirement.

An application view defines:

- **Communication with the EIS**, including connection settings, login credentials, and so on.
- **Service invocations**, including the information that the EIS requires for the request, as well as any service request and response schemas associated with the service.
- **Event notifications**, including the information that the EIS publishes and the event schemas for inbound messages.

You create application views in either of two ways:

- Using the Application View Design Console. For detailed information about application views, see "What Are Application Views?" in "Understanding Application Integration" in *Using Application Integration* at the following URL:

  `http://edocs.bea.com/wli/docs81/aiover/2intfra.html`

- Writing custom code. For more information, see "Using Application Views by Writing Custom Code" in *Using Application Integration* at the following URL:

  `http://edocs.bea.com/wli/docs81/aiuser/4usrcust.html`

An application view for the Adapter for MQSeries provides these features:

- Standards-based data representation. All events, requests, and responses are represented as or encapsulated in standards-based XML.
- Abstraction from the details of the EIS. Application views offer a level of abstraction from the details of the underlying EIS, freeing the developers to concentrate on the business processes and data and not on the configuration and details of that system.

To learn more about application views, see Chapter 2, "Defining Application Views for MQSeries."

Service Clients and Event Consumers

In an integration solution, there are clients that invoke services and consumers for event notifications.

Service Clients

A variety of clients can invoke services on an EIS via an application view. They include BEA WebLogic Workshop business processes, web services, and portals; queries and BEA Liquid Data; and custom Java applications.

For more information, see the following topics in the BEA WebLogic Workshop Help System:

- "Building Integration Applications"
- "Building Web Services"
- "Building Portal Applications"

In addition, see "Using Applications With Business Processes" in *Using the Application Integration Design Console* at the following URL:

`http://edocs.bea.com/wli/docs81/aiuser/3usruse.html`

Event Consumers

Adapters deliver events using the WebLogic Integration Message Broker, which provides business processes with a channels-based publish and subscribe communication mechanism. Consumers can include BEA WebLogic Workshop business processes, web services, and portals, as well as custom Java applications.

For more information, see the following topics in the BEA WebLogic Workshop Help System:

- "Message Broker Subscription Control" in "Message Broker Controls"
- "Building Enterprise Applications"

at the following URL:

`http://edocs.bea.com/workshop/docs81/doc/en/core/index.html`

In addition, see "Receiving Events" in "Using Applications With Business Processes" in *Using the Application Integration Design Console* at the following URL:

`http://edocs.bea.com/wli/docs81/aiuser/3usruse.html`

About the BEA WebLogic Adapter for MQSeries

The BEA WebLogic Adapter for MQSeries connects to your MQSeries system so that you can easily use your MQSeries data and functions within your business processes. The adapter provides scalable, reliable, and secure access to your MQSeries system.

This section includes the following topics:

Supported MQSeries Operations for Application Integration

The Adapter for MQSeries supports application integration by sending and receiving data as messages that allow business applications to exchange information across platforms. The adapter provides the following functions:

- Synchronous and asynchronous, bi-directional message interactions between WebLogic Integration and MQSeries managed queues.

- Data transfer between a business process running within WebLogic Integration and an MQSeries Queue Manager.
- Services and events for end-to-end business process management using XML schemas.

Supported Services

The Adapter for MQSeries supports four types of services: Transaction, SendMessage, SendRequest, and GetMessage. In each case, the service accepts an XML request document as an input and executes the service and generates an XML response document as output.

These are the services supported by the Adapter for MQSeries:

- Transaction allows you to create and manage a transaction scope for the remaining services to be executed within a workflow.
- SendMessage sends a datagram or reply message to a specific queue.
- SendRequest sends a request message to a specific queue.
- GetMessage gets a message from a specified queue.

Supported Events

The Adapter for MQSeries has a single event. The event is configured to receive messages from a particular queue, and is triggered when messages arrive at the queue. The event then generates an XML response document for each message.

Benefits of the Adapter for MQSeries

The combination of the adapter and WebLogic Integration supplies everything you need to integrate your WebLogic Integration business processes and enterprise applications with your MQSeries system. The Adapter for MQSeries provides these benefits:

- Integration can be achieved without custom coding.
- Business processes can be started by events generated by MQSeries.
- Business processes can request and receive data from your MQSeries system using services.

- Adapter events and services are standards-based. The adapter services and events provide extensions to the *J2EE Connector Architecture* (JCA) version 1.0 from Sun Microsystems, Inc. For more information, see the Sun JCA page at the following URL:

  `http://java.sun.com/j2ee/connector/`

- The adapter and WebLogic Integration solution is scalable. The BEA WebLogic Platform provides clustering, load balancing, and resource pooling for a scalable solution. For more information about scalability, see the following URL:

  `http://e-docs.bea.com/wls/docs81/cluster/index.html`

- The adapter and WebLogic Integration solution benefits from the fault-tolerant features of the BEA WebLogic Platform. For more information about high availability, see the following URL:

  `http://edocs.bea.com/wli/docs81/deploy/index.html`

- The adapter and WebLogic Integration solution is secure, using the security features of the BEA WebLogic Platform and the security of your MQSeries system. For more information about security, see the following URL:

  `http://edocs.bea.com/wls/docs81/secintro/index.html`

Getting Started With the Adapter for MQSeries

This section gives an overview of how to get started using the BEA WebLogic Adapter for MQSeries within the context of an application integration solution. Integration with MQSeries involves the following tasks:

- Step 1: Design the Application Integration Solution
- Step 2: Determine the Required MQSeries Business Workflows
- Step 3: Define Application Views and Configure Services and Events
- Step 4: Integrate with Other BEA Software Components
- Step 5: Deploy the Solution to the Production Environment

Step 1: Design the Application Integration Solution

The first step is to design an application integration solution, which includes (but is not limited to) such tasks as:

- Defining the overall scope of application integration.

- Determining the business process(es) to integrate.
- Determining which WebLogic Platform components will be involved in the integration, such as web services, business processes, or portals designed in the WebLogic Workshop environment.
- Determining which external systems and technologies will be involved in the integration, such as MQSeries systems and other EISs.
- Determining which BEA WebLogic Adapters will be required, such as the BEA WebLogic Adapter for MQSeries. An application integration solution can involve multiple adapters.

This step involves the expertise of business analysts, system integrators, and EIS specialists (including MQSeries specialists). Note that an application integration solution can be part of a larger integration solution.

To learn more about designing an application integration solution, see *Designing WebLogic Integration Solutions* at the following URL:

`http://edocs.bea.com/wli/docs81/design/index.html`

Step 2: Determine the Required MQSeries Business Workflows

Within the larger context of an application integration project, you must determine which queues, queue managers, and connection channels you need to access in MQSeries to support the business processes in the application integration solution.

Factors to consider include (but are not limited to):

- Queue managers, queues, and connection channels used to access MQSeries.
- MQSeries transactions involved in business processes
- Logins required to access MQSeries transports and perform the required operations
- Whether operations are, from the adapter point of view:
  - services, which notify MQSeries, via an MQSeries XML document, with a request for action, and, in addition, whether such services should be processed synchronously or asynchronously
  - events, which are notifications from MQSeries that trigger workflows

This step usually involves the expertise of MQSeries specialists, including analysts and administrators.

Step 3: Define Application Views and Configure Services and Events

You should create an application view that provides an XML-based interface between WebLogic Integration (WLI) and a particular MQSeries system within your enterprise. If you are accessing multiple MQSeries queue managers, you define a separate application view for each MQSeries system you want to access. To provide different levels of security access (such as "guest" and "administrator"), define a separate application view for each security level.

Once you define an application view, you can configure events and services in that application view. To learn more about defining application views, see Chapter 2, "Defining Application Views for MQSeries" in conjunction with *Using the Application Integration Design Console*, at the following URL:

`http://edocs.bea.com/wli/docs81/aiuser/index.html`

Step 4: Integrate with Other BEA Software Components

Once you have configured and published one or more application views for MQSeries integration, you can integrate these application views into other BEA software components, such as business processes or portals built in the BEA WebLogic Workshop environment.

For more information, see *Using the Application Integration Design Console*, particularly the topic, "Using Application Views with Business Processes," at the following URL:

`http://edocs.bea.com/wli/docs81/aiuser/index.html`

Step 5: Deploy the Solution to the Production Environment

After you have designed, built, and tested your application integration solution, you can deploy it into a production environment. The following list describes some of the tasks involved in deploying an application integration:

- Design the deployment.
- Deploy the required components of the BEA WebLogic Platform.
- Install and deploy the BEA WebLogic Adapter for MQSeries as described in *BEA WebLogic Adapter for MQSeries Installation and Configuration Guide*
- Deploy your application views for MQSeries integration.
- Verify business processes in the production environment.
- Monitor and tune the deployment.

To learn more about deploying your application integration solution, see *Deploying WebLogic Integration Solutions* at the following URL:

`http://edocs.bea.com/wls/docs81/deploy/index.html`

CHAPTER 2

Defining Application Views for MQSeries

An application view is a business-oriented interface to objects and operations within an EIS.

This section presents the following topics:

How to Use This Document

This document is designed to be used in conjunction with *Using the Application Integration Design Console*, available at the following URL:

`http://edocs.bea.com/wli/docs81/aiuser/index.html`

Using the Application Integration Design Console describes, in detail, the process of defining an application view, which is a key part of making an adapter available to process designers and other users. What *Using the Application Integration Design Console* does *not* cover is the specific information—about connections to your MQSeries system, as well as supported services and events—that you must supply as part of the application view definition. You will find that information in this section.

At each point in *Using the Application Integration Design Console* where you need to refer to this document, you will see a note that directs you to a section in your adapter user guide, with a link to the edocs page for adapters. The following road map illustration shows where you need to refer to this document from *Using the Application Integration Design Console*.

Figure 2-1 Information Interlock with *Using the Application Integration Design Console*

Before You Begin

Before you define an application view, make sure you have:

- Installed and deployed the adapter according to the instructions in *BEA WebLogic Adapter for MQSeries Installation and Configuration Guide*.
- Determined which business processes need to be supported by the application view. The required business processes determine the types of services and events you include in your application views. Therefore, you must gather information about the application's business requirements from the business analyst. Once you determine the necessary business

processes, you can define and test the appropriate services and events. For more information, see "Getting Started With the Adapter for MQSeries" on page 1-9.

- Gathered the connection information for your MQSeries system.

About Application Views

An application view defines:

- Connection information for the EIS, including login information, connection settings, and so on.
- Service invocations, including the information the EIS requires for this request, as well as the request and response schemas associated with the service.
- Event notifications, including the information the EIS publishes and the event schema for inbound messages.

Typically, an application view is configured for a single business purpose and contains only the services and events required for that purpose. An EIS might have multiple application views, each defined for a different purpose.

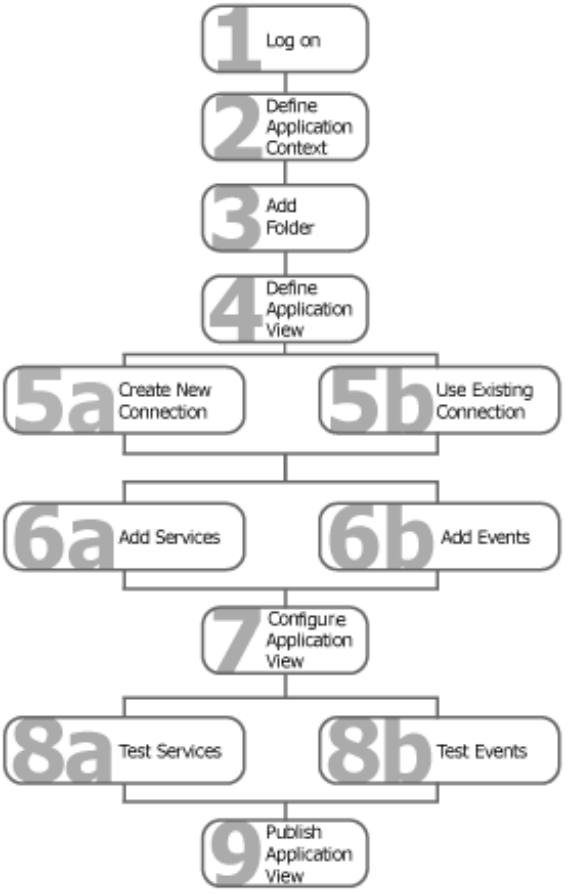
About Defining Application Views

Defining an application view is a multi-step process described in *Using the Application Integration Design Console*, available at the following URL:

`http://edocs.bea.com/wli/docs81/aiuser/index.html`

The information you enter depends on the requirements of your business process and your EIS system configuration. Figure 2-2 summarizes the procedure for defining and configuring an application view.

Figure 2-2 Process for Defining and Configuring an Application View



To define an application view:

1. Log on to the WebLogic Integration Application View Console.
2. Define the application context by selecting an existing application or specifying a new application name and root directory.

   This application will be using the events and services you define in your application view. The application view works within the context of this application.
3. Add folders as required to help you organize application views.

4. Define a new application view for your adapter.
5. Add a new connection service or select an existing one.

   If you are adding a new connection service, see "Defining MQSeries Connection Parameters" on page 2-5 for details about MQSeries requirements.
6. Add the events and services for this application view.

   See the following sections for details about MQSeries requirements:

   – "Setting Service Properties" on page 2-12

   – "Setting Event Properties" on page 2-19
7. Perform final configuration tasks.

   If you are adding an event connection, see "Defining Event Connection Parameters" on page 2-23 for details about MQSeries requirements.
8. Test all services and events to make sure they can properly interact with the target MQSeries system.

   See the following sections for details about MQSeries requirements:

   – "Testing Services" on page 2-25

   – "Testing Events Manually" on page 2-28
9. Publish the application view to the target WebLogic Workshop application.

   This is the application you specified in step 2. Publishing the application view allows business processes within the target application to interact with the newly published application view using an Application View control.

Note: After you publish the application view, it is good practice to stop testing it. This ensures that only one application view is deployed on WebLogic Server.

Defining MQSeries Connection Parameters

1 2 3 4 **5** 6 7 8 9

This information applies to "Step 5A, Create a New Browsing Connection" in *Using the Application Integration Design Console*, at the following URL:

`http://edocs.bea.com/wli/docs81/aiuser/index.html`

The Select Browsing Connection page allows you to choose the type of connection factory to associate with the application view. You can select an existing connection or create a connection.

Connections are contained within an adapter instance. The Adapter for MQSeries provides two types of connection settings: Local Bindings connection and TCP/IP connection.

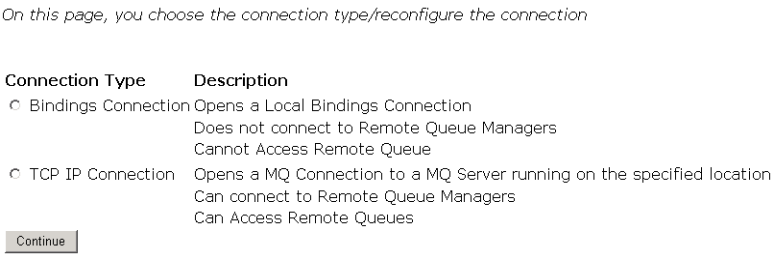
You can create a new browsing connection in one of the following ways:

- Configuring a Bindings Connection
- Configuring a TCP/IP Connection

To create a new browsing connection:

1. In the Define New Application View page, enter the application view name and description, select the associated adapter, and click Create New Connection. The Create New Browsing Connection page appears.
2. In the Create New Browsing Connections page, enter a connection name and description, as described in *Using the Application Integration Design Console*.
3. Click Define. The Select the Type of Connection page displays descriptions of the two connection types.

   Select the required connection type.

   – To configure a bindings connection, see "Configuring a Bindings Connection" on page 2-7.

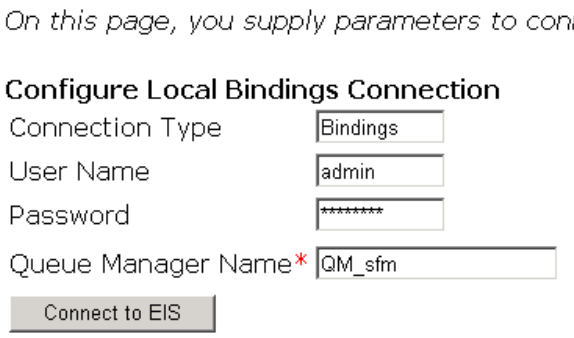
   – To configure a TCP/IP connection, see "Configuring a TCP/IP Connection" on page 2-8.

Configuring a Bindings Connection

The Bindings Connection provides connection to the MQSeries Server running on the system where the adapter is installed.

To configure a Bindings connection:

1. On the Select the Type of Connection page, select Bindings Connection and click Continue. The Configure Connection Parameters page appears, displaying the fields required to establish a Local Bindings Connection.



Note: A red asterisk (*) indicates that a field is required.

2. Enter the following information:

Table 2-1 Local Bindings Connection Parameters

Parameter	Description
User Name	Your WebLogic user name
Password	Your WebLogic password
Queue Manager Name	The name of the Queue Manager

3. Click Connect to EIS.

   You return to the Create New Browsing Connections page, where you can specify connection pool parameters and logging levels. For more information, see *Using the Application Integration Design Console* at the following URL:

   http://edocs.bea.com/wli/docs81/aiuser/index.html

Configuring a TCP/IP Connection

A TCP/IP connection allows you to connect to any MQSeries Server that is running on the same network where the adapter is installed.

This section contains the following topics:

Implementing User Exits

If you are using a TCP/IP connection, you can also implement user exits for processing messages or performing security operations. The available exits are:

- Send Exit: users can implement logic to process the message before sending it to a queue.
- Receive Exit: users can implement logic to process the message after retrieving it from a queue.
- Security Exit: users can implement logic to perform security operations, such as authentication.

The corresponding Java MQSeries exit interfaces that should be implemented are:

- MQSendExit
- MQReceiveExit
- MQSecurityExit

You can configure user exits when you configure a TCP/IP connection for the adapter. At design-time, you can choose one or more of the available exits. For each exit you select, you must provide an ExitAppName, which is the fully qualified name of the class file that implements the corresponding Java MQSeries Exit interface. The class file you specify must be available in the CLASSPATH of the application server where the Adapter for MQSeries is deployed.

For more information about implementing user exits, see your MQSeries documentation.

To implement a user exit:

1. Create a user exit class as follows:

```
package com.bea.UserExit;
import com.ibm.mq.MQSendExit;
public class UserSendExit implements MQSendExit {
public UserSendExit()
{
}
public byte[] sendExit(MQChannelExit channelExit,MQChannelDefinition
channelDefnition,byte[] agentBuffer)
{
// Your code goes here
return agentBuffer;
}
}
```

   The name of this User Exit is UserSendExit. Similarly, for a Receive Exit, the class should implement `MQReceiveExit`, and for a Security Exit, the class should implement `MQSecurityExit`.

2. Compile this class and create a `JAR` file for the class file, for example `senduserexit.jar`.

3. Set the CLASSPATH to include this `JAR` file.

   To know how to set the `CLASSPATH`, see *BEA WebLogic Adapter for MQSeries Installation and Configuration Guide* at

   `http://edocs.bea.com/wladapters/mq/docs81/index.html`

4. To use this exit, you specify its the fully qualified class name in the Configure TCP/IP Connection page.

   For example, `com.bea.UserExit.UserSendExit`.

   For more information about TCP/IP connection settings, see "Setting TCP/IP Parameters" on page 2-10.

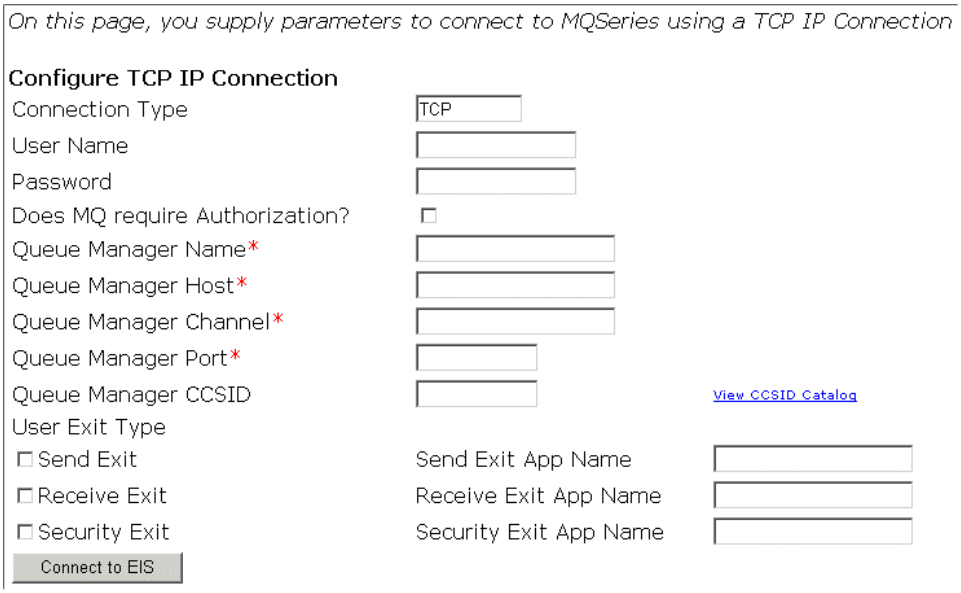
Setting TCP/IP Parameters

A TCP/IP connection allows you to connect to any MQSeries Server that is running on the same network where the adapter is installed.

To set the TCP/IP parameters:

1. On the Select the type of connection page, select TCP/IP Connection and click Continue.

   The Configure Connection Parameters page displays the fields required for establishing a TCP/IP connection.



   Note: A red asterisk (*) indicates that a field is required.

2. Enter the following information:

Table 2-2 TCP/IP Connection Parameters

Parameter	Description
User Name	Your WebLogic user name
Password	Your WebLogic password

Table 2-2 TCP/IP Connection Parameters (Continued)

Parameter	Description
Does MQ Require Authorization?	If MQSeries requires an authorization, this setting lets you use the same user name and password to connect to it
Queue Manager Name	The name of the MQSeries queue manager
Queue Manager Host	The name of the host where the MQSeries Server is installed
Queue Manager Channel	The name of the server connection channel to connect to the queue manager
Queue Manager Port	The port number of the queue manager
Queue Manager CCSID	The coded character set ID for the character set used by the queue manager to establish the connection and process message headers. This should be a valid integer that corresponds to the character set to be used. For example, 1208, 819. **Note:** Be sure to select a CCSID that is supported by the MQ Server's operating system. To see the CCSID catalog, click the View CCSID Catalog link. For more information on CCSID, see your MQSeries documentation or contact the MQ Administrator.
User Exit Type	The exit types (if required) and their corresponding implementation classes to process a message while it is sent or received from a queue, or to perform security operations on the message. **Note:** The corresponding exit implementation class must be created and made available before you can use them here.

3. Click Connect to EIS.

   You return to the Create New Browsing Connections page, where you can specify connection pool parameters and logging levels. For more information, see *Using the Application Integration Design Console* at the following URL:

   http://edocs.bea.com/wli/docs81/aiuser/index.html

Setting Service Properties

1 2 3 4 5 **6** 7 8 9

This information applies to "Step 6A, Add a Service to an Application View" in *Using the Application Integration Design Console*, at the following URL:

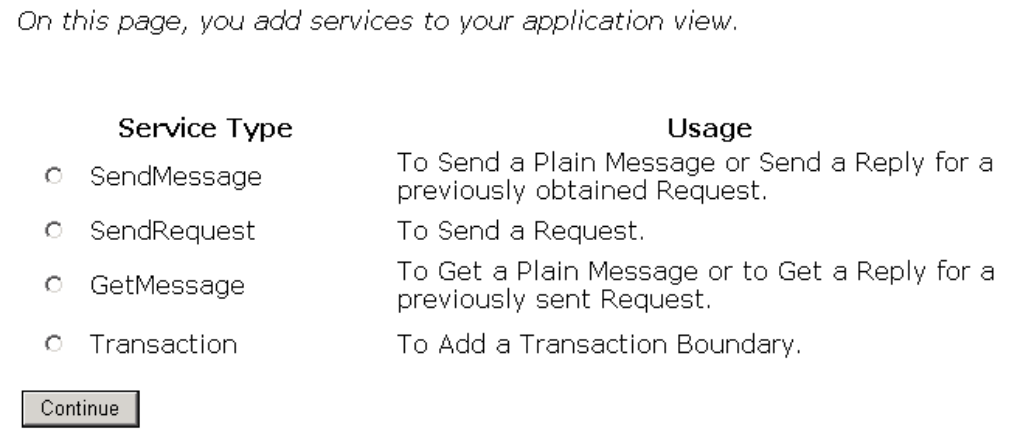
`http://edocs.bea.com/wli/docs81/aiuser/index.html`

Adapter for MQSeries uses services to make requests of the MQSeries system. A service consists of both a request and a response. The Adapter for MQSeries supports the following services:

- Transaction Service
- SendMessage and SendRequest Service
- GetMessage Service

To configure services:

1. On the Application View Administration page, click Add in the Services row. The Add Service page appears.



2. Select the Service Type and click Continue. The selected service page appears.

Transaction Service

This information applies to "Step 6A, Add a Service to an Application View" in *Using the Application Integration Design Console*, at the following URL:

`http://edocs.bea.com/wli/docs81/aiuser/index.html`

The transaction service in the Adapter for MQSeries exposes MQSeries Syncpoint capability to the invoking client application. Syncpoints are MQSeries units of work. This capability allows clients to invoke operations within a unit of work or, in other words, to define the transaction scope of services. The adapter provides two types of transaction services:

- Implicit Transaction

  This type of transaction does not require the user to create and manage a transaction scope to execute services. It provides the option of executing a Send or a Get service within a specific transaction scope. The Adapter for MQSeries internally creates and manages the transaction for such a transaction scope, freeing the user from the necessity to do so.

- Explicit Transaction

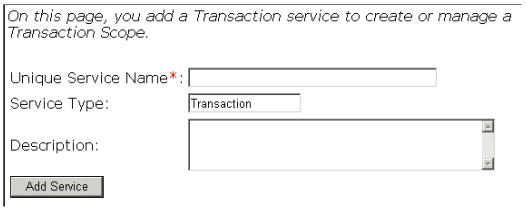
  This is the default transaction type. You can use an explicit transaction service in a business process to begin and end a transaction scope. The transaction scope starts with a Transaction Begin, can contain as many services as required in the middle, and ends with either a Transaction Commit or a Transaction BackOut, depending on the process requirements. The services invoked between the Transaction Begin and the Transaction Commit or BackOut are part of the transaction scope.

  Note: An explicit transaction scope is not required for services that use the Implicit Transaction setting.

  See "Using a Transaction Service in a Process" on page 3-2 to know how to use a transaction service in a WebLogic Integration business process.

To configure a Transaction Service:

1. On the Add Service page, select Transaction and click Continue. The Transaction page appears.



Note: A red asterisk (*) indicates that a field is required.

2. Enter a unique service name that describes the function the service performs and a brief description of the service.
3. Click Add Service. The service is added to the application view.

SendMessage and SendRequest Service

This information applies to "Step 6A, Add a Service to an Application View" in *Using the Application Integration Design Console*, at the following URL:

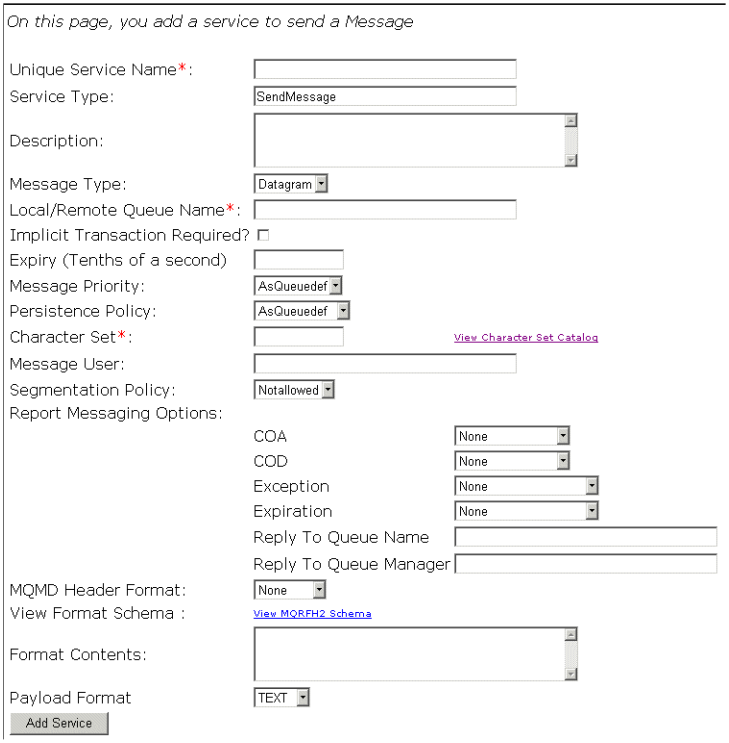
`http://edocs.bea.com/wli/docs81/aiuser/index.html`

A SendMessage service sends either a Datagram Message or a Reply Message to a queue. A SendRequest service sends a Request message to a queue.

To configure a SendMessage or a SendRequest service:

1. On the Add Service page, select SendMessage or SendRequest and click Continue.

   The corresponding page appears. This is the SendMessage page. In the SendRequest page, Message Type is preset to Request and Reply to Queue Name is required.



Note: A red asterisk (*) indicates that a field is required.

2. Enter a unique service name that describes the function the service performs and a description for the service.
3. For SendMessage services only, select the message type. The choices are:
   - Datagram: a message sent to a single destination with no reply expected
   - Reply: a response to a request message received earlier

   For SendRequest service, the message type is Request. This type of message expects a reply.
4. Enter the name of the local or remote queue to which the message has to be sent.

   For a Binding connection, before you can send messages to a remote queue, you must configure the following:

– a remote queue definition, a transmission queue, and a sender channel on the queue manager to which you are connected.

– a local queue and a receiver channel on the remote queue manager.

For a TCP/IP connection, you need a Server Connection Channel, and you must make sure that the MQSeries listener is configured.

Note: MQSeries does not support getting messages from remote queues.

5. If an implicit transaction is required, turn on the Implicit Transaction setting.

   The default is explicit transaction. An implicit transaction does not require the user to create a transaction scope for this service execution. Instead, the adapter creates the transaction scope. For more information about implicit and explicit transactions, see "Implicit Transaction" on page 2-13 and "Explicit Transaction" on page 2-13.

6. Enter the length of time the message should remain on the queue waiting to be picked up, in tenths of a second. Leave the field blank or specify a value of -1 for an unlimited expiry.

   The message will remain in the queue for the specified time, after which it will expire and be deleted from the queue.

7. Select the message's priority level. The lowest priority is 0, and the highest priority is 9.

   If you select the default value, AsQueuedef, the message inherits the priority level defined in the queue.

8. Select the persistence policy, as follows.

   The persistence policy determines whether messages remain in the queue after a system restart.

Table 2-3 Persistence Policy Settings

Parameter	Description
Persistent	Messages are recovered and remain in the queue
NotPersistent	Messages are no longer available on the queue when MQ Server restarts
AsQueuedef	Messages have the persistence policy defined in the queue

9. Enter a valid character set. Click the Character Set Catalog link to view the available character sets.

This is the CCSID to which outgoing message data will be converted.

10. Enter the name of the MCA message user.

    This user should be a member of the MQSeries Administration group for reports to be delivered to the specified ReplyTo Queue.

11. Select a segmentation policy.

    This policy determines whether a message can be segmented before sending it to the queue. If the setting is Allowed, the queue manager can segment the message according to the criteria set in the queue. If the setting is NotAllowed, the message cannot be segmented.

12. Select the report messaging options to receive reports on the messages sent.

    Each of the four report types can be set to no report, or to contain no data, some data (the first 100 bytes of the message) or full data (the full content of the message).

Table 2-4 Report Messaging Parameters

Parameter	Description
COA	Sends a confirmation of arrival report to the queue configured to receive reports.
COD	Sends a confirmation of delivery report to the queue configured to receive reports.
Exception	Sends an exception report to the queue configured to receive reports, if an exception occurs while sending the message to the queue.
Expiration	Sends an expiration report to the queue configured to receive reports, after the message in the queue expires.

Note: COD, Exception, and Expiration reports require an MQ Server authorization. Unless the message user is authorized by the MQSeries Server, messages will be delivered to the dead letter queue on the queue manager.

13. Enter a Reply to Queue Name.
    - For SendMessage services, enter the name of the queue for report messages.
    - For SendRequest services, enter the name of the queue where replies are expected (required).

14. Enter a Reply to Queue Manager

    This is the name of the queue manager for report messages.

15. Select the header format.

    If the message is in a string format, select String. If the message requires an MQRFH2 header format, select MQRFH2. If the message does not require any header format, select None.

16. If you have selected MQRFH2 as the header format, enter the header contents using the format of the MQRFH2 schema. Click the link to view the MQRFH2 schema and use it to develop the contents.

17. Select the Payload format of the message: TEXT, Binary, or XML.

18. Click Add Service. The service is added to the application view.

GetMessage Service

1 2 3 4 5 **6** 7 8 9

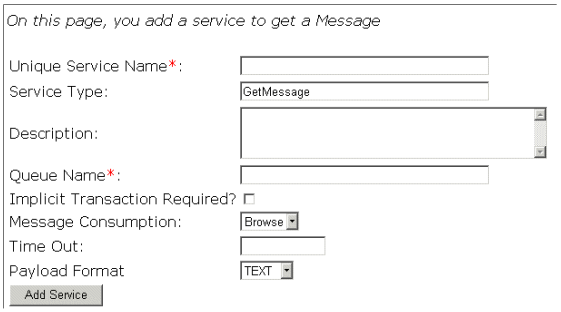
This information applies to "Step 6A, Add a Service to an Application View" in *Using the Application Integration Design Console*, at the following URL:

`http://edocs.bea.com/wli/docs81/aiuser/index.html`

A GetMessage service receives a message from a message queue.

To configure a GetMessage service:

1. On the Add Service page, select GetMessage and click Continue. The GetMessage page appears.



Note: A red asterisk (*) indicates that a field is required.

2. Enter a unique service name that describes the function the service performs, and a description for the service.
3. Enter the following information:

Table 2-5 GetMessage Service Parameters

Parameter	Definition
Queue Name	Name of the queue from where the message should be received. **Note:** MQSeries does not support getting messages from remote queues.
Implicit Transaction	A transaction that does not require the user to create a transaction scope for this service execution. The adapter takes care of the transaction scope for this type of service. The default is explicit transactions. See "Implicit Transaction" on page 2-13 for details.
Message Consumption	• Browse (default): lets you browse messages without deleting them. Messages remain intact on the queue after they are read. If this option is used, the MQ Administrator must periodically remove outdated and unwanted messages from the queue. • Delete: used to delete a message from the queue after it is read.
Time Out	The length of time, in seconds, to await receipt of a message. Leave this field blank, or set it to -1 to wait indefinitely.
Payload Format	The payload format applicable to the message to be received. The default is TEXT.

4. Click Add Service. The service is added to the application view.

Setting Event Properties

1 2 3 4 5 **6** 7 8 9

This information applies to "Step 6B, Add an Event to an Application View" in *Using the Application Integration Design Console*, at the following URL:

`http://edocs.bea.com/wli/docs81/aiuser/index.html`

An event monitors a specified queue to receive messages that match configuration information you provide when you define the event. The event creates an event response document for the message retrieved. The event response document describes the MQMD properties and the message data.

Messages that cause errors (also known as Poison messages) while the event response document is being generated are moved to the error queue that you specify when you configure the event. If no error queue is specified, messages remain in their original queue, but no response is generated. The error queue can be a local or remote queue. For more information on remote queues, see your MQSeries documentation.

This section contains the following topics:

Setting up Content Filtering

Using the content filtering feature in an event, you can filter the messages in a queue based on their contents. To set this option, you specify an optional payload filter, which consists of a string of filtering characters, including case-sensitive wildcards.

To set content filtering for an event:

1. Extract the `contentfilter.jar` from the `BEA_MQSERIES_8_1.ear` file and include it in the CLASSPATH of the environment where the Content Filtering Code will be developed.
2. Create a Content Filter class by extending the following Class:
   `com.bea.adapter.mqseries.AbstractContentFilter`

   Note: This class is present in the `contentfilter.jar` file that you extracted in step 1.
3. Write the Code for the Content Filter Class as follows:

```
package com.bea.adapter.mqseries.contentfilter;
import com.bea.adapter.mqseries.utils.AbstractContentFilter;
public class ContentFilter extends AbstractContentFilter{
public ContentFilter ()
{
}
public boolean matchContent(byte[]message)
{
boolean isMatching = false;
//Filtering logic to go here. isMatching to be made 'true' if the message
meets the filtering logic.
return isMatching;
}
```

   Note: The `matchContent` function is abstract within the `com.bea.adapter.mqseries.utils.AbstractContentFilter` class. To

incorporate your logic for filtering messages based on their contents, you should override the `matchContent` function within the content filter class.

4. Compile this class and create a `JAR` with a name, for example, `mycontentfilter.jar`.
5. Include this `JAR` file in the CLASSPATH of the Start Server script. For more information on setting the `CLASSPATH`, see "Extract the Adapter Files and Change the WebLogic Startup Script" in *BEA WebLogic Adapter for MQSeries Installation and Configuration Guide*.
6. Start the WebLogic Server.
7. While configuring the event in the application view, select the Content Filtering Required option.
8. Give the fully qualified class name of the Content Filter. For example, `com.bea.adapter.mqseries.contentfilter.ContentFilter`.
9. Add the event and deploy the application view.

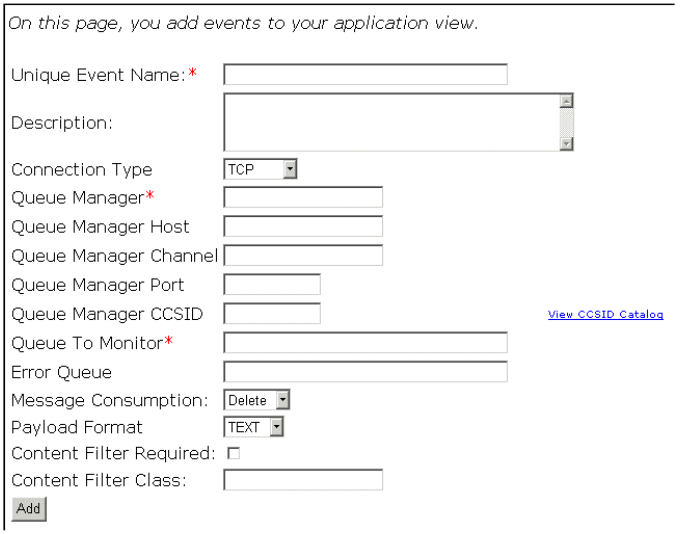
When the event is executed, the message obtained from the queue is passed to the `matchContent` function as an input. Depending upon the Boolean value returned by this function, an event response is generated for the message. This is in the form of an XML document describing the MQMD attributes of the message and the application data.

Configuring Event Parameters

An event monitors a specified queue to receive messages that match configuration information you provide when you define the event. The event creates an event response document for the message retrieved. The event response document describes the MQMD properties and the message data.

To configure an event:

1. On the Application View Administration page, click the Add button in the Events row. The Event page appears.



Note: A red asterisk (*) indicates that a field is required.

2. Enter a unique event name that describes the function the event performs, and a description for the event.

3. Enter the following information:

Table 2-6 Event Parameters

Parameter	Description
Connection Type	The type of connection to the MQ Server (TCP or Bindings)
Queue Manager	The name of the queue manager
Queue Manager Host	The name of the host where the queue manager resides
Queue Manager Channel	The name of the queue manager channel
Queue Manager Port	The port number where the queue manager is running
Queue Manager CCSID	The coded character set ID (CCSID) for the character set used by the queue manager to establish a connection and process message headers. This is also the CCSID of the payload of incoming messages. This should be a valid integer that corresponds to the character set to be used. Click the View CCSID Catalog link to view a list of links.

Table 2-6 Event Parameters (Continued)

Parameter	Description
Queue to Monitor	The name of the queue to monitor for messages. This should be a local queue.
Error Queue	The name of the queue for storing poison messages
Message Consumption	• Browse: allows for browsing messages without deleting them. Messages remain intact on the queue after they are read. If this option is used, the MQ Administrator must periodically remove outdated and unwanted messages from the queue. • Delete (default): deletes a message from the queue after it is read.
Payload Format	The data format applicable to the message received. The default is TEXT.
Content Filter Required	Filters the contents of the message received.
Content Filter Class	The class file for the content filter. Required if content filter is selected.

4. Click Add. The event is added to the application view.
5. See "Defining Event Connection Parameters" on page 2-23 for information about setting connection parameters for an event.

Defining Event Connection Parameters

This information applies to "Step 7, Perform Final Configuration Tasks" in *Using the Application Integration Design Console*, at the following URL:

`http://edocs.bea.com/wli/docs81/aiuser/index.html`

Once you have finished adding services and events and have saved your application view, you must perform some final configuration tasks, including configuring event delivery connections, before testing the services and events. You perform these configuration tasks from the Final Configuration and Testing page.

To define event connection parameters:

1. In the Connections area on the Application View Administration page, click Select/Edit.

The Connection Information for Application View page appears.

2. In the Event Connection area, click the Select Existing link.

   The Select Event Connection for Application View page appears.

3. Select an Event connection from the available adapter instances, and click OK.

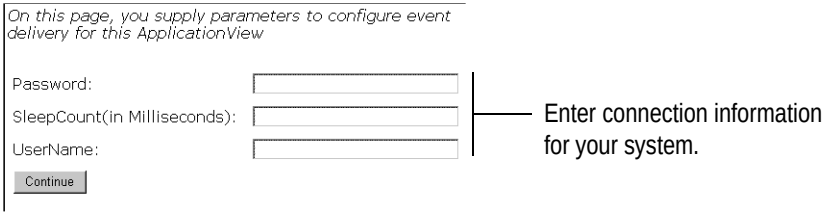
   You return to the Connection Information for Application View page.

4. Under Event Connection, select the Event link.

   The Edit Event Connection page appears.

5. In the Connection Parameters area, click Define.

   The Configure Event Delivery Parameters page appears.



6. Enter the following information:

Table 2-7 Event Connection Parameters

Parameter	Description
Password	The password for your WebLogic Server Administration Console user name.
SleepCount	The number of milliseconds the adapter will wait between polling for events.
User Name	Your WebLogic Server Administration Console user name, defined in the `startWebLogic` script.

The event delivery parameters you enter on this page enable connection to your MQSeries system and are used when generating events.

7. Click Continue to return to the Edit Event Connection page.

8. Set the log configuration.

The Edit Event Adapter page allows you to define event parameters and configure the information that will be logged for the connection factory. Select one of the following settings for the log:

- Log errors and audit messages
- Log warnings, errors, and audit messages
- Log informational, warning, error, and audit messages
- Log all messages (maximum tracing level)

The table that follows describes the type of information that each logging message contains.

Table 2-8 Logging message categories

This type of message	Contains
Audit	Extremely important information related to the business processing performed by an adapter.
Error	Information about an error that has occurred in the adapter, which may affect system stability.
Warning	Information about a suspicious situation that has occurred. Although this is not an error, it could have an impact on adapter operation.
Information	Information about normal adapter operations.

9. Click OK.

   You return to the Connection Information for Application View page, where you can click Back to return to the Application View Administration page.

Testing Services

This information applies to "Step 8A, Test an Application View's Services" in *Using the Application Integration Design Console*, at the following URL:

`http://edocs.bea.com/wli/docs81/aiuser/index.html`

The purpose of testing an application view service is to evaluate whether that service interacts properly with the target MQSeries system. When you test a service, you supply any inputs

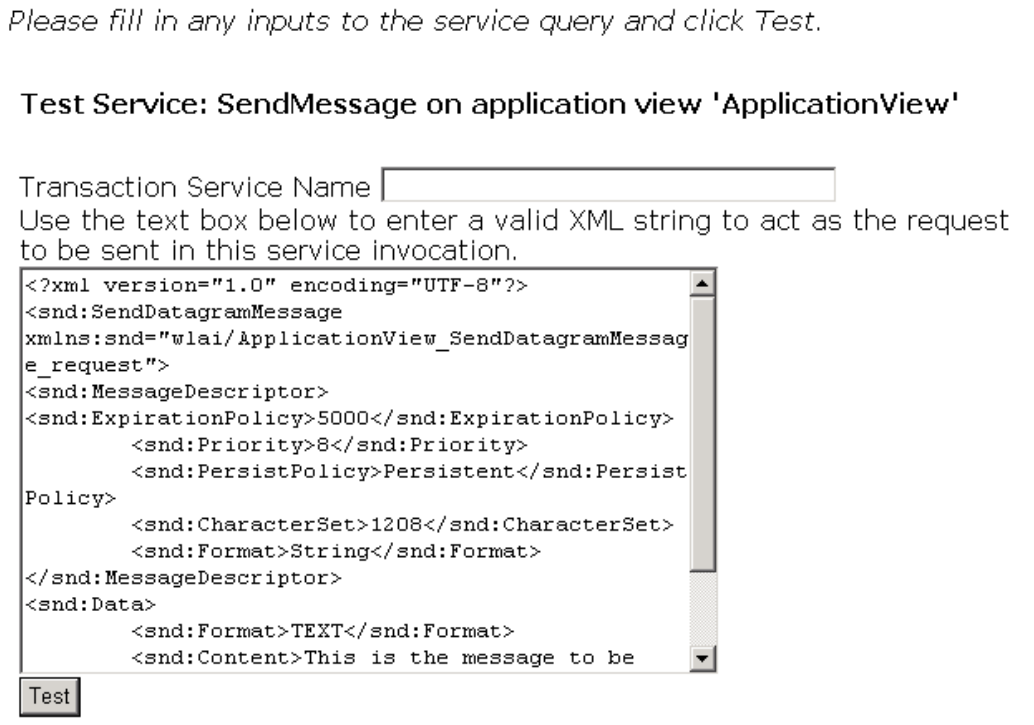
required to start the service. For the Adapter for MQSeries, the input is in the form of a valid XML document that acts as input for the service.

Note: Before you test an application view, it must contain at least one event or service. Also, you must place the application view in test mode. To place an application view in test mode, click the Test button at the bottom of the Application View Administration page.

To test a service:

1. In the Summary for Application View page, click the Test link beside the service to be tested.

   The Test Services page appears.



2. Enter the Transaction Name only if you have not selected the Implicit Transaction option, while configuring the service during design-time.
3. Enter the sample request document that matches the request schema for the selected service. For example,

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<snd:SendDatagramMessage
xmlns:snd="wlai/ApplicationView_SendDatagramMessage_request">
<snd:MessageDescriptor>
<snd:ExpirationPolicy>5000</snd:ExpirationPolicy>
<snd:Priority>8</snd:Priority>
<snd:PersistPolicy>Persistent</snd:PersistPolicy>
<snd:CharacterSet>1208</snd:CharacterSet>
<snd:Format>String</snd:Format>
</snd:MessageDescriptor>
<snd:Data>
<snd:Format>TEXT</snd:Format>
<snd:Content>This is the message to be put into the queue</snd:Content>
</snd:Data>
</snd:SendDatagramMessage>
```

4. Click Test.

   The Test Results page displays the XML Response document generated after a successful execution.

This page shows the results from testing a service.

Input to service SendDatagramMessage on application view ApplicationView

```
<snd:MessageDescriptor>
<snd:ExpirationPolicy>5000</snd:ExpirationPolicy>
        <snd:Priority>8</snd:Priority>
        <snd:PersistPolicy>Persistent</snd:Persist
Policy>
        <snd:CharacterSet>1208</snd:CharacterSet>
        <snd:Format>String</snd:Format>
</snd:MessageDescriptor>
<snd:Data>
        <snd:Format>TEXT</snd:Format>
        <snd:Content>This is the message to be
put into the queue</snd:Content>
</snd:Data>
</snd:SendDatagramMessage>
```

Output from service SendDatagramMessage on application view ApplicationView

```
<?xml version="1.0"?>
<ns0:SendDatagramMessage
xmlns:ns0="wlai/ApplicationView_SendDatagramMessag
e_response">
  <ns0:Result>

<ns0:MessageId>414D5120514D5F6974706C5F30323530E64
50A3F12600000</ns0:MessageId>
    <ns0:Status>SUCCESS</ns0:Status>
    <ns0:Error/>
  </ns0:Result>
</ns0:SendDatagramMessage>
```

Execution time: 50192 (ms)

Testing Events Manually

This information applies to "Step 8B, Test an Application View's Events" in *Using the Application Integration Design Console*, at the following URL:

http://edocs.bea.com/wli/docs81/aiuser/index.html

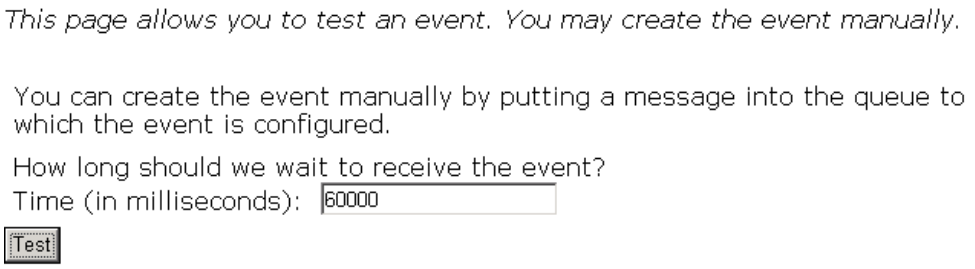
The purpose of testing an application view event is to make sure that the adapter correctly handles events.

Note: Before you test an application view, it must contain at least one event or service. Also, you must place the application view in test mode. To place an application view in test mode, click the Test button at the bottom of the Application View Administration page.

To test an event manually:

1. In the Summary for Application View page, click the Test link beside the event to be tested.

   The Test Event page appears.



2. In the Time field, enter a reasonable period of time to wait, specified in milliseconds, before the test times out (One second = 1000 milliseconds. One minute = 60,000 milliseconds.).
3. Click Test. The test waits for an event to trigger it. One of the following is shown as result:
   – If the test succeeds, the Test Result page displays the event document generated by the event.

This page shows the results from testing a event.

Generated event of type Event on application view ApplicationView

```
<?xml version="1.0"?>
<ns:Event
xmlns:ns="wlai/ApplicationView_Event_event">
  <ns:EventInfo>
    <ns:QueueName>MessageQueue</ns:QueueName>

<ns:MessageConsumption>browse</ns:MessageConsumpti
on>
  </ns:EventInfo>
  <ns:PayLoad>
    <ns:MQMD>
      <ns:MessageType>Datagram</ns:MessageType>

<ns:MessageId>414D5120514D5F6974706C5F30323530E645
0A3F32410000</ns:MessageId>
```

Execution time: 734 (ms)

– If the test fails or if the event is not generated in the allotted time, the Test Result page displays a Timed Out message.

CHAPTER 3

Using the Adapter for MQSeries

This section describes how to use MQSeries services and events in a process. It contains information on the run-time requirements for executing services and events, and the request and response documents involved in executing a process.

The information provided here assumes that you have in-depth knowledge of the WebLogic Integration capabilities in WebLogic Workshop 8.1. To learn more about executing business processes in the WebLogic Workshop environment, see the WebLogic Workshop and WebLogic Integration documentation at the following URLs:

`http://edocs.bea.com/workshop/docs81/doc/en/core/index.html`

`http://edocs.bea.com/wli/docs81/doc/index.html`

This section contains the following topics:

Using a Transaction Service in a Process

A Transaction service in a process invokes a set of services as part of a transaction scope. The transaction service provides three options:

- Transaction Begin: begins a transaction scope with the associated MQ connection.
- Transaction Commit: ends a transaction scope and save the changes made since the beginning of the transaction scope.
- Transaction BackOut: ends a transaction scope and rolls back the changes made since the beginning of the transaction scope.

You invoke a service with a transaction scope. Transaction Begin is the first service in the transaction scope, and Transaction Commit or Transaction BackOut is the last. In between, you can invoke as many services as you need—iteratively within the transaction scope, and in any sequence. This type of transactions is known as an explicit transaction, as described in "Explicit Transaction" on page 2-13.

You can configure a service with an implicit transaction, which means that you can execute the service independently in a process without defining a transaction scope. For more information, see "Implicit Transaction" on page 2-13.

The following illustration shows an application view called mqCust that has three defined services.

Figure 3-1 Sample Application View with Services Defined

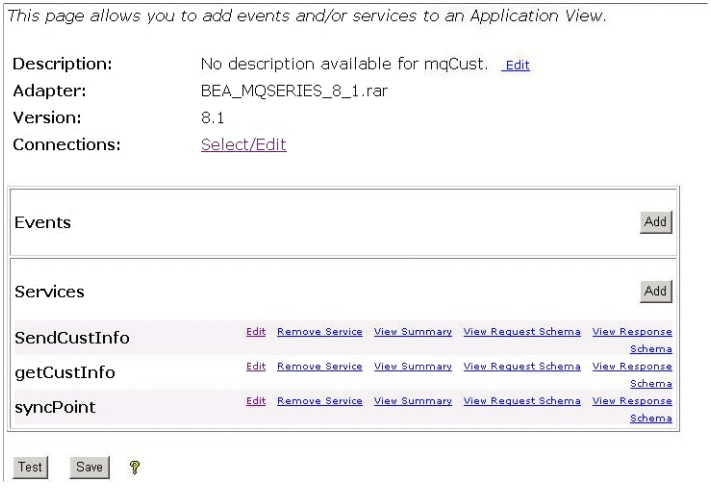
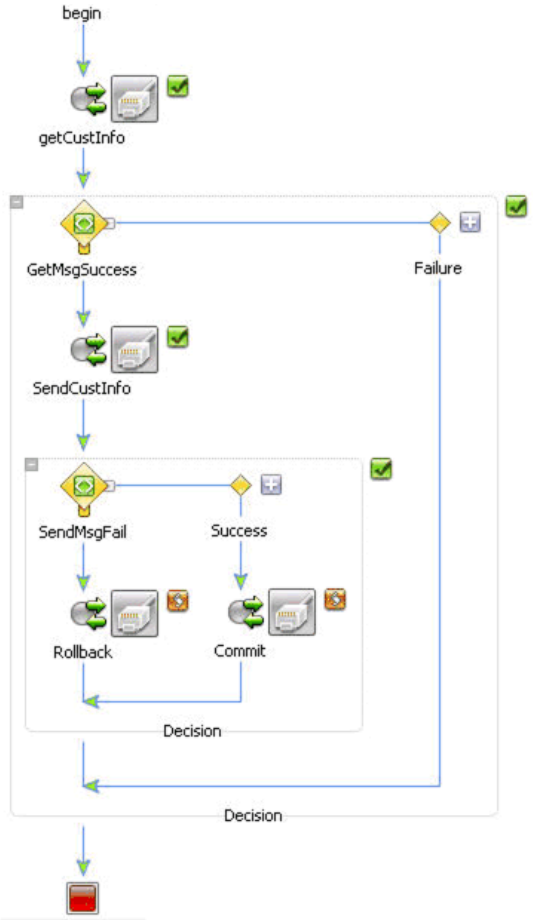


Figure 3-2 demonstrates how to use the mqCust services in a process. This process gets a message from an MQSeries queue, copies the contents of the message, and puts it onto another MQSeries queue. This is an explicit transaction scope that starts an MQSeries syncpoint operation and ends with a Transaction Commit. It does error checking and calls a Transaction BackOut if the operation to put a message on the queue fails. Figure 3-3 shows the logic for error checking.

Figure 3-2 Process with Adapter for MQSeries Transaction Scope



Begin calls the syncpoint Transaction service to start an MQSeries syncpoint operation. This Transaction service is defined in the mqCust application view. The request document identifies it as a Transaction-Begin. This is the start of the transaction scope.

getCustInfo calls getCustInfo, a GetMessage service that is also defined in the mqCust application view.

GetMsgSuccess checks to see if the return status of the MQSeries getCustInfo operation is successful. If so, the process continues. If not, the process takes the Failure path.

SendCustInfo sends the message to the MQSeries queue. It also copies the contents of the message received in getCustInfo to this message. There is a simple transformation step in the Send Data section of this node.

SendMsgFail checks the status of SendCustinfo. If the step failed, the process takes the path to RollBack. If the step succeeded, the process takes the path that leads to the Commit.

RollBack calls the syncpoint Transaction service, this time with a request document that identifies it as a Transaction-BackOut service.

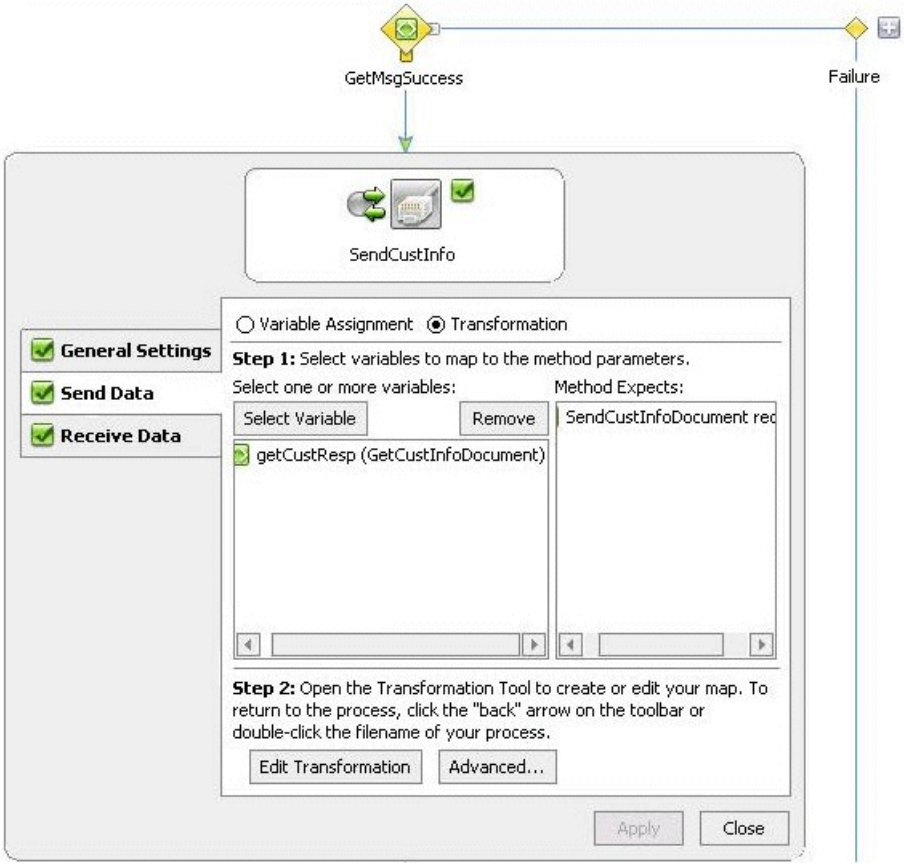
Commit also calls the syncpoint Transaction service, with a request document that identifies it as a Transaction-Commit service. The transaction scope ends with either Commit or RollBack.

For each service you must provide a request document that is formatted in accordance with the corresponding schema. You can override message descriptor values and data format, and provide application data for Send services. The process illustrated above requires a total of five request

documents, one for each service invocation. For more information about the formats for each document, see Appendix A, "Request and Response Schemas."

The following illustration shows the SendCustInfo node, which includes a transformation step.

Figure 3-3 SendCustInfo Node Transformation



After successful execution, a response document is generated for each service based on the response schema of that service. See Appendix A, "Request and Response Schemas," to view the schemas for each service.

Note: The request documents for each service contain some tags to which appropriate values should be set at run time. See Appendix B, "Run-Time Parameter Values," for information on the tags for each service.

About Creating Request Documents

A request document is an XML document that the Adapter for MQSeries uses when it executes services. Request documents can contain MQ message descriptor values, header information, and application data.

This section describes how you can override descriptor values and how to provide header information. It includes these topics:

- Overriding MQ Message Descriptor Attributes
- Providing MQRFH2 Information

Overriding MQ Message Descriptor Attributes

You can set MQ Message Descriptor (MQMD) attributes when you configure services and events. The request documents for SendRequest and SendMessage services have a MessageDescriptor tag, which contains the following tags:

Table 3-1 Message Descriptor Attributes

Tag	Description
ExpirationPolicy	Determines the length of time the message should remain on the queue waiting to be picked up, in tenths of a second.
Priority	Sets the message's priority level. The lowest priority is 0 and the highest priority is 9. With a setting of AsQueuedef, the message inherits the priority level defined in the queue.
PersistPolicy	Determines whether messages remain in the queue after a system restart. Possible values are: • Persistent: Messages are recovered and remain in the queue. • NotPersistent: Messages are no longer available on the queue when it restarts. • AsQueuedef: Messages have the persistent policy defined in the queue.
CharacterSet	Identifies the character set used by the queue manager to establish the connection and process message headers. This should be a valid integer that corresponds to the character set to be used. For example, 1208 or 819.

Table 3-1 Message Descriptor Attributes

Tag	Description
Format	Determines the format used for message data. The choices are None, String, or MQRFH2.

These tags correspond to fields in the SendMesage and SendRequest configuration pages. At run-time, the values that you set for these tags will override the values specified for the service.

Note: If any of the MQMD values are either set incorrectly or omitted from the service configuration or the request document, the adapter will use the default values.

For a list of default MQMD values, see your MQSeries documentation.

Providing MQRFH2 Information

The MQSeries Rules and Formatting Header 2 (MQRFH2) is a message header that provides information about the structure and formatting of the message content that follows. In queues serviced by a broker, MQRFH2 information speeds the sending of messages.

When you create a SendMessage or SendRequest request document, you can include MQRFH2 information in the following format:

```
<MQRFH2>
        <Encoding></Encoding>
        <CodedCharSetId></CodedCharSetId>
        <Format></Format>
        <NameValueCCSID></NameValueCCSID>
        <NameValueDatan></NameValueDatan>
</MQRFH2>
```

You can also view the MQRFH2 schema when you configure a SendMessage or SendRequest service.

Creating Request Documents

This section contains information on setting values in the request documents for sending different types of messages to meet specific requirements.

The sample request documents in this section use the service name as the Root element. This is not mandatory. The adapter accepts any well-formed document with any Root element name and

valid values. The adapter also supports request and response documents without a namespace. A response document is generated by the application for each service. See "About Response Documents" on page 3-18 for the sample response documents.

Note: You can view the request and response schema for each service on the Summary for Application View page.

This section contains the following topics:

SendMessage Datagram Request Document

You can send a Datagram message using the SendMessage service. Before sending a Datagram message, do the following in the SendMessage request document:

- Configure the Message Descriptor data that you want to override. For details, see "Overriding MQ Message Descriptor Attributes" on page 3-5.
- Provide the `MQRFH2` data if applicable. See "Providing MQRFH2 Information" on page 3-6.
- In the Data tag, do the following:
  - Provide the required format: TEXT, Binary, or XML.
  - Provide the application data that you want to send to the queue (required). The application data tag should not be empty.

You can use the same SendMessage service to send as many Datagram messages as the destination queue can hold. For more information about the destination queue's message holding capacity, contact your MQ Server Administrator.

Sample SendMessage Request Document

These are the setting for a sample SendMessage request document:

Table 3-2 Sample SendMesage Request Document Values

Tag	Value
ExpirationPolicy	5000
Priority	8
PersistPolicy	Persistent
CharacterSet	1208
Format	TEXT
Application data	`This is the message to be put into the queue`

This is the corresponding sample SendMessage request document:

```
<?xml version="1.0" encoding="UTF-8"?>
<snd:SendDatagramMessage
xmlns:snd="wlai/ApplicationView_SendDatagramMessage_request">

<snd:MessageDescriptor>
<snd:ExpirationPolicy>5000</snd:ExpirationPolicy>
       <snd:Priority>8</snd:Priority>
       <snd:PersistPolicy>Persistent</snd:PersistPolicy>
       <snd:CharacterSet>1208</snd:CharacterSet>
       <snd:Format>String</snd:Format>
</snd:MessageDescriptor>

<snd:Data>
       <snd:Format>TEXT</snd:Format>
       <snd:Content>This is the message to be put into the
queue</snd:Content>

</snd:Data>

</snd:SendDatagramMessage>
```

See "SendMessage Response Document" on page 3-19 for the corresponding sample response document.

SendMessage Reply Request Document

You can send a Reply message using a SendMessage service that is configured with a message type of Reply. Before sending a reply message, do the following in the SendMessage request document:

- Configure the Message Descriptor data that you want to override. For details, see "Overriding MQ Message Descriptor Attributes" on page 3-5.
- Provide the `MQRFH2` data if applicable. See "Providing MQRFH2 Information" on page 3-6.
- Provide the `CorrelationId` of the Reply message (required).
- Provide an optional `MessageId`. If you do not provide the `MessageId`, the `MessageId` of the reply message is generated by the queue manager. This ID is a hexadecimal String.
- In the Data tag, provide the following:
  - The required format: TEXT, Binary, or XML.
  - The application data that you want to send to the queue (required). The application data tag should not be empty.

You can use the same SendMessage service to send as many Reply Messages as the destination queue can hold. For more information about the destination queue's message holding capacity, contact your MQ Server Administrator.

Sample SendMessage Reply Request Document

```
<?xml version="1.0" encoding="UTF-8"?>

<snd:SendReplyMessage
xmlns:snd="wlai/ApplicationView_SendReplyMessage_request">

<snd:MessageDescriptor>

 <snd:ExpirationPolicy>5000</snd:ExpirationPolicy>

 <snd:Priority>8</snd:Priority>

 <snd:PersistPolicy>Persistent</snd:PersistPolicy>

 <snd:CharacterSet>1208</snd:CharacterSet>

 <snd:Format>String</snd:Format>

</snd:MessageDescriptor>
```

```
<snd:MessageId>414D5120514D5F6974706C5F303235301882173F12500600
</snd:MessageId>
<snd:CorrelationId>414D5120514D5F6974706C5F3032353070A0163F12800100
</snd:CorrelationId>
<snd:Data>
 <snd:Format>TEXT</snd:Format>
 <snd:Content>hello world</snd:Content>
</snd:Data>
</snd:SendReplyMessage>
```

SendRequest Request Document

You can send a Request message using the SendRequest service. Before sending a Request message, do the following in the SendRequest request document:

- Configure the Message Descriptor data that you want to override. For details, see "Overriding MQ Message Descriptor Attributes" on page 3-5.
- Provide the `MQRFH2` data if applicable. See "Providing MQRFH2 Information" on page 3-6.
- In the Data tag, do the following:
  - Supply the required format: TEXT, Binary, or XML.
  - Provide the application data that you want to send to the queue (required). The application data tag should not be empty.

You can use the same SendRequest service to send as manyRequest messages as the destination queue can hold. For more information about the destination queue's message holding capacity, contact your MQ Server Administrator.

Sample SendRequest Request Document

```
<?xml version="1.0" encoding="UTF-8"?>
<snd:SendRequestMessage
xmlns:snd="wlai/ApplicationView_SendRequestMessage_request">
<snd:MessageDescriptor>
<snd:ExpirationPolicy>5000</snd:ExpirationPolicy>
<snd:Priority>8</snd:Priority>
```

```
<snd:PersistPolicy>Persistent</snd:PersistPolicy>
<snd:CharacterSet>1208</snd:CharacterSet>
<snd:Format>String</snd:Format>
</snd:MessageDescriptor>

<snd:Data>

<snd:Format>TEXT</snd:Format>
<snd:Content>This is the message to be put into the queue</snd:Content>
</snd:Data>

</snd:SendRequestMessage>
```

See "SendRequest Response Document" on page 3-19 for the corresponding sample response document.

GetMessage Request Document

You can receive messages from a specified queue using the GetMessage service. To receive a message, set values for one or more of the following tags in the GetMessage request document:

- MessageId: The MessageId of the message to be received. This is optional.
- CorrelationId: The CorrelationId of the message to be received. This is optional.
- GroupId: The GroupId of the message to be received. This is optional.

  Note: If you want to receive a group message, ensure that the `GroupId` tag is present in the request document. The first message in the queue bearing the specified `GroupId` is received.

- DataFormat: The data format of the message to be received: TEXT, Binary, or XML.

A message that matches these given IDs is received. If none of these values is provided, the first message in the queue is received.

Note: When you configure a GetMessage service for an XML data format, the service will throw a `javax.resource.ResourceException` if the message received is in TEXT or Binary data format.

Sample GetMessage Request Document

```
<?xml version="1.0" encoding="UTF-8"?>

<gmn:GetTEXTMessage
xmlns:gmn="wlai/ApplicationView_GetTEXTMessage_request">
```

```
    <gmn:MessageId>414D5120514D5F6974706C5F30323530A4F63F3B12B00100</gmn:
MessageId>
    <gmn:DataFormat>TEXT</gmn:DataFormat>
</gmn:GetTEXTMessage>
```

See "GetMessage Response Document" on page 3-20 for the corresponding sample response document.

Transaction Request Documents

There are three different request documents, one for each type of transaction service: Transaction Begin, Transaction Commit, and Transaction BackOut. A transaction scope begins with Transaction Begin and ends with either Transaction Commit or Transaction BackOut.

This section lists the formats for the following Transaction Service request documents:

- Transaction Begin Request Document
- Transaction Commit Request Document
- Transaction BackOut Request Document

See "Transaction Response Document" on page 3-19 for a sample response document.

Transaction Begin Request Document

```
<?xml version="1.0" encoding="UTF-8"?>
<tr:TransactionService
xmlns:tr="wlai/ApplicationView_TransactionService_request">
  <tr:TransactionFunction>Begin</tr:TransactionFunction>
</tr:TransactionService>
```

Transaction Commit Request Document

```
<?xml version="1.0" encoding="UTF-8"?>
<tr:TransactionService
xmlns:tr="wlai/ApplicationView_TransactionService_request">
  <tr:TransactionFunction>Commit</tr:TransactionFunction>
</tr:TransactionService>
```

Transaction BackOut Request Document

```
<?xml version="1.0" encoding="UTF-8"?>

<tr:TransactionService
xmlns:tr="wlai/ApplicationView_TransactionService_request">

  <tr:TransactionFunction>BackOut</tr:TransactionFunction>
```

</tr:TransactionService>

Using Data Formats in Services and Events

The Adapter for MQSeries supports TEXT, Binary, and XML data formats. Although you configure a service or event with a data format when you create it, you can override that data format in the SendMessage, SendRequest, and GetMessage services in the service's request document. You cannot override the configured data format for an event.

Table 3-3 Data Formats for Services and Events

For This Data Format	And This Type of Document	The Content Tag Data Should Be
TEXT	Request document or Response document	Plain text
Binary or XML	Request document or Response document	Base64 encoded

The following is an example showing how to send an XML application data in a Base64 encoded format. Each element is in its own line (a requirement for well-formed XML):

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
<header>hello world</header>
<body>this is a test message</body>
</data>
```

This XML application data should be Base64 encoded and put as value within the `Content` tag of the request document. The `Format` tag data should be XML.

The request document containing Base64 encoded XML application data within the `Content` tag is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<ServiceName>
<MessageDescriptor>
       <ExpirationPolicy>5000</ExpirationPolicy>
       <Priority>8</Priority>
       <PersistPolicy>Persistent</PersistPolicy>
       <CharacterSet>813</CharacterSet>
</MessageDescriptor>

<Data>
<Format>XML</Format>
<Content>
PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz48ZGF0YT48Ym9keT50aGlzIG
lzIGEgdGVzdCBtZXNzYWdlPC9ib2R5PjwvZGF0YT4=
</Content>
</Data>
</ServiceName>
```

Note: The `ServiceName` tag can be any valid tag name. The request document must be well formed and comply with the XML schema for the service to execute properly. If the XML application data (the Payload) was not well formed prior to Base64 encoding, or if it is not included in the request document, the application will throw an exception and the service will be terminated. You can follow the same process to send a message with Binary data format.

Working with Group Messages

A group message is a set of messages that share a common GroupId. Each message in the group has a unique Message Sequence Number (MsgSeqNumber). The MsgSeqNumber for the first message in the group is 1 and is sequentially incremented for each subsequent message in the group. You must provide the GroupId and MsgSeqNumber as inputs while sending group messages. You can do this in two ways:

- You can let the queue manager generate the GroupId and MsgSeqNumber for the first message, and then add logic to the business process to assign the GroupId and MsgSeqNumbers for all subsequent messages. See "Sending Group Messages Using a GroupId Generated by the Queue Manager" on page 3-16 for more information.
- You can provide a GroupId and MsgSeqNumber for all the messages in the group, including the first message. See "Sending Group Messages Using a User-Specified GroupId" on page 3-17 for more information.

Note: When you assign a GroupId, make sure that it is unique for this group within the queue to which the messages are sent.

Group Message Request Schema

The Adapter for MQSeries lets you send Group Messages using SendMessage and SendRequest services. The request schema of these services has the following tags that are used to send Group messages.

```
<xsd:element name="GroupMessageOptions" minOccurs="0">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="IsFirstMessage" type="xsd:boolean" minOccurs="0"/>
<xsd:element name="IsLastMessage" type="xsd:boolean" minOccurs="0"/>
<xsd:element name="IsIntermediateMessage" type="xsd:boolean"
minOccurs="0"/>
<xsd:element name="GroupId" type="xsd:string" minOccurs="0"/>
<xsd:element name="MsgSeqNumber" type="xsd:string" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

Group Message Option Tags

The GroupMessageOptions tag is normally optional in the request document, but it becomes required when you send Group Messages.

While sending Group Messages you should make the following settings depending upon the sequence of messages you send:

Table 3-4 Group Message Tags

For This Message	Set This Tag Value to True	Omit Values for These Tags
First	`IsFirstMessage`	`IsLastMessage` and `IsIntermediateMessage`
Intermediate	`IsIntermediateMessage`	`IsFirstMessage` and `IsLastMessage`
Last	`IsLastMessage`	`IsFirstMessage` and `IsIntermediateMessage`

For the first message in the group, the `GroupId` is optional, but if you provide a value, it should be the same GroupId used for all subsequent group messages in the queue. If you don't provide a value, the MQ Server generates a unique GroupID and MsgSeqNumber and returns them to the user in the response document. The MsgSeqNumber is usually 1 for the first message in the group.

For the intermediate and last messages, the `GroupId` is required. It should be the same value that was returned in the response document after the first message was sent. If it is not the same value, the MQ Server will create a new group message. The MsgSeqNumber is also required and should be the incremented value of the MsgSeqNumber that was returned in the response document of the previous message in the group.

For more information on Group message concepts, see your MQSeries documentation.

Sending Group Messages Using a GroupId Generated by the Queue Manager

If you use the GroupId generated by the MQSeries Server, you do not have to provide the GroupId and MsgSeqNumber for the first message in the group. Instead, you can allow the queue manager to generate the GroupId and MsgSeqNumber for the first message. However, you must supply additional logic in the process to get the GroupId and MsgSeqNumber from the first message's response document and set the GroupId as input for the subsequent messages in the group. You must also provide the logic that picks up the MsgSeqNumber and increments it correctly for the subsequent messages.

To send group messages using a GroupID generated by the queue manager:

1. Determine the type of message that you want to send, create an application view, and add the required services. For explicit transactions, add a Transaction service to start and stop the transaction scope during execution. For more information, see "About Defining Application Views" on page 2-3.
2. Deploy the application view.
3. Create a process using WebLogic Workshop. See BEA WebLogic Workshop Help at the following URL:

   `http://edocs.bea.com/workshop/docs81/doc/en/core/index.html`

4. Provide additional logic in the process to pick the GroupId and MsgSeqNumber from the first message's response document and set the same GroupId as input in the subsequent service's request documents. Each MsgSeqNumber should be an incremented value of the previous message's MsgSeqNumber.

Note: Be sure to omit the GroupId and MsgSeqNumber for the first message in the group.

5. Create a request document that corresponds to each message in the group.

6. Execute the process. For each invocation of the SendMessage or SendRequest service, provide the corresponding request document as input. The group messages are sent to the specified queue.

Sending Group Messages Using a User-Specified GroupId

You can specify the GroupId and MsgSeqNumber in the GroupId and MsgSeqNumber tags in the request document for all messages in the group. Be sure that within the queue, the GroupId used is unique for the group.

To send the group messages using a user-specified GroupID:

1. Determine the type of message that you want to send, create an application view, and add the required services. For explicit transactions, add a Transaction service to start and stop the transaction scope during execution. For more information, see "About Defining Application Views" on page 2-3.

2. Deploy the application view.

3. Create a process using WebLogic Workshop. See BEA WebLogic Workshop Help at the following URL:

   `http://edocs.bea.com/workshop/docs81/doc/en/core/index.html`

4. Provide a GroupId and MsgSeqNumber. Be sure that the GroupId is unique for this group and that it is used for both the first message and all subsequent messages. Set the MsgSeqNumber for the first message to 1, and increment it by 1 for each subsequent message.

5. Create a request document that corresponds to each message in the group.

6. Execute the process. For each invocation of the SendMessage or SendRequest service, provide the corresponding request document as input. The group messages are sent to the specified queue.

Receiving Group Messages

You can receive group messages from the queue using a GetMessage service or an event. The GetMessage service retrieves only the first message with the specified ID in the queue. To retrieve all messages in the group, you must invoke the GetMessage service iteratively with a message consumption setting of 'Delete' and the GroupID in the request document. In an event, all the messages in the queue, including group messages, are received.

About Response Documents

The request and response schemas for Adapter for MQSeries support the usage of Namespaces. However the adapter also supports request documents without NameSpaces. In either case, the response documents for services and events are generated with a default Namespace.

The adapter generates an event document for each event you create. This document has a MQMD section, which contains the message descriptors, and a data section, which contains the application data and header information.

The sample response documents in this section use the service name as the Root element. This is not mandatory. The adapter generates a response document for each service. See "About Creating Request Documents" on page 3-5 for the sample request documents.

Note: You can view the request schema and the response schema for each service on the Summary for Application View page.

This section includes the following sample response documents:

- Transaction Response Document
- SendMessage Response Document
- SendRequest Response Document
- GetMessage Response Document
- Event Response Document

Transaction Response Document

```
<?xml version="1.0" encoding="UTF-8"?>

<ns0:TransactionService xmlns:ns0="wlai/
ApplicationView_TransactionService_response">

<ns0:Result>

<ns0:Status>SUCCESS</ns0:Status>

<ns0:Error/>

</ns0:Result>

</ns0:TransactionService>
```

SendMessage Response Document

```
<?xml version="1.0"?>

<ns0:SendDatagramMessage
xmlns:ns0="wlai/ApplicationView_SendDatagramMessage_response">

<ns0:Result>

<ns0:MessageId>414D5120514D5F6974706C5F30323530E6450A3F12600000</ns0:
MessageId>

<ns0:Status>SUCCESS</ns0:Status>

<ns0:Error/>

<ns0:GroupMessage/>

</ns0:Result>

</ns0:SendDatagramMessage>
```

SendRequest Response Document

```
<?xml version="1.0"?>

<ns0:SendRequestMessage

xmlns:ns0="wlai/ApplicationView_SendRequestMessage_response">

<ns0:Result>
```

```
<ns0:MessageId>414D5120514D5F6974706C5F30323530E6450A3F12B00000
</ns0:MessageId>

<ns0:Status>SUCCESS</ns0:Status>

<ns0:Error/>

<ns0:GroupMessage/>

</ns0:Result>

</ns0:SendRequestMessage>
```

GetMessage Response Document

```
<?xml version="1.0"?>

<ns0:GetTEXTMessage
xmlns:ns0="wlai/ApplicationView_GetTEXTMessage_response">

  <ns0:Result>

    <ns0:Status>SUCCESS</ns0:Status>

    <ns0:Error/>

    <ns0:GetInfo>

      <ns0:QueueName>MessageQueue</ns0:QueueName>

      <ns0:MessageConsumption>Browse</ns0:MessageConsumption>

    </ns0:GetInfo>

    <ns0:PayLoad>

      <ns0:MQMD>

        <ns0:MessageType>Datagram</ns0:MessageType>
<ns0:MessageId>414D5120514D5F6974706C5F30323530E6450A3F12900000</ns0:
MessageId>

        <ns0:CorrelationId></ns0:CorrelationId>
<ns0:GroupId>414D5120514D5F6974706C5F30323530E6450A3F22900000</ns0:
GroupId>

        <ns0:Format>MQSTR   </ns0:Format>

       <ns0:ReplyToQueueName>ReportQueue
</ns0:ReplyToQueueName>
```

```
      <ns0:ReplyToQueueManagerName>QM_itpl_025051
</ns0:ReplyToQueueManagerName>
        <ns0:UserId>admin       </ns0:UserId>
        <ns0:ApplicationIdData>
</ns0:ApplicationIdData>
        <ns0:PutApplicationName>MQSeries Client for Java
</ns0:PutApplicationName>
        <ns0:PutDateTime>9/7/2001 - 6:13:11</ns0:PutDateTime>
        <ns0:ApplicationOriginData>    </ns0:ApplicationOriginData>
      </ns0:MQMD>
      <ns0:Message>
        <ns0:MQRFH2_Contents/>
        <ns0:Data>
          <ns0:Content>This is the message to be put into the
queue</ns0:Content>
        </ns0:Data>
      </ns0:Message>
    </ns0:PayLoad>
  </ns0:Result>
</ns0:GetTEXTMessage>
```

Event Response Document

```
<?xml version="1.0"?>
<ns:EventforTEXT xmlns:ns="wlai/ApplicationView_EventforTEXT_event">
  <ns:EventInfo>
    <ns:QueueName>MessageQueue</ns:QueueName>
    <ns:MessageConsumption>browse</ns:MessageConsumption>
  </ns:EventInfo>
  <ns:PayLoad>
```

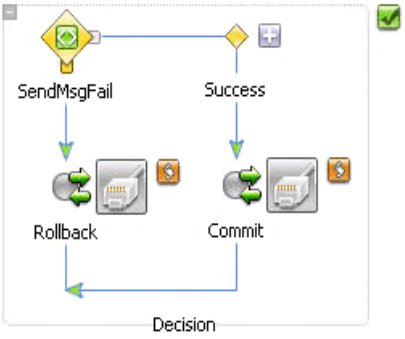
```
    <ns:MQMD>
      <ns:MessageType>Request</ns:MessageType>
<ns:MessageId>414D5120514D5F6974706C5F30323530E6450A3F12B00000</ns:
MessageId>
      <ns:CorrelationId></ns:CorrelationId>
      <ns:GroupId></ns:GroupId>
      <ns:Format>MQSTR   </ns:Format>
      <ns:ReplyToQueueName>ReplyQueue</ns:ReplyToQueueName>
<ns:ReplyToQueueManagerName>QM_itpl_025051</ns:ReplyToQueueManagerName>
      <ns:UserId></ns:UserId>
      <ns:ApplicationIdData></ns:ApplicationIdData>
      <ns:PutApplicationName>MQSeries Client for
Java</ns:PutApplicationName>
      <ns:PutDateTime>9/7/2001 - 6:23:19</ns:PutDateTime>
      <ns:ApplicationOriginData></ns:ApplicationOriginData>
    </ns:MQMD>
    <ns:Message>
      <ns:MQRFH2_Contents/>
      <ns:Data>This is the message to be put into the queue</ns:Data>
    </ns:Message>
  </ns:PayLoad>
</ns:EventforTEXT>
```

Handling Errors and Exceptions

If an MQ exception occurs during the execution of a service, the adapter builds a response document with status set to FAILURE. Additionally, an error tag displays the MQ reason code for the exception, the completion code, and the MQ description for the MQ reason code.

For all other exceptions, the adapter throws a `javax.resource.ResourceException` back to the invoking application. It is the client application's responsibility to call a Transaction BackOut to roll back the changes that had occurred since the start of the current Transaction Scope. This

should be done to safeguard the application data. The illustration that follows shows the process node with a decision that invokes a Transaction-BackOut or a Transaction-Commit, depending on the status of the services in the transaction scope.



Transaction Scope Error Conditions

If a transaction scope already exists before you execute a Transaction service with Transaction-Begin, the Transaction service invocation fails and the adapter generates a response document with the status tag set to FAILURE. When any other service is invoked and a transaction scope does not exist, the adapter throws a `javax.resource.ResourceException`, and the service invocation is terminated.

If a transaction scope exists, but the service execution is not successful, the adapter throws a `javax.resource.ResourceException`, and the service invocation is terminated. If the service execution is successful, the user can invoke a Transaction-Commit to save the process changes permanently.

If a ResourceException is thrown during process execution, the user can invoke a Transaction-BackOut to roll back the changes that happened since the start of the current transaction scope.

APPENDIX A

Request and Response Schemas

This section contains the request and response schemas generated by the Adapter for MQSeries for services and events. The request documents you create must be in accordance with the corresponding request schema. Likewise, the response document generated by the adapter will be in accordance with the adapter's response schema. These schemas are generated by the adapter when you configure services and events.

These are the request and response schemas generated by the Adapter for MQSeries:

- Transaction Request Schema
- Transaction Response Schema
- SendMessage Request Schema
- SendMessage Response Schema
- SendRequest Request Schema
- SendRequest Response Schema
- GetMessage Request Schema
- GetMessage Response Schema
- Event Schema

To view the schemas, click the schema link (request and response) on the Application View Administration page of the Application Integration Design Console. The supported schemas follow.

Transaction Request Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="wlai/ApplicationView_TransactionService_request"
elementFormDefault="qualified"
targetNamespace="wlai/ApplicationView_TransactionService_request">

<xsd:element name="TransactionService">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="TransactionFunction" type="xsd:string"/>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:schema>
```

Transaction Response Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="wlai/ApplicationView_TransactionService_response"
elementFormDefault="qualified"
targetNamespace="wlai/ApplicationView_TransactionService_response">

<xsd:element name="TransactionService">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Result">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Status" type="xsd:string"/>

<xsd:element name="Error">
```

```
<xsd:complexType>
<xsd:sequence>
<xsd:element name="CompletionCode" type="xsd:string" minOccurs="0"/>
<xsd:element name="ReasonCode" type="xsd:string" minOccurs="0"/>
<xsd:element name="ReasonCodeDescription" type="xsd:string"
minOccurs="0"/>
<xsd:element name="Trace" type="xsd:string" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

SendMessage Request Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="wlai/ApplicationView_SendDatagramMessage_request"
elementFormDefault="qualified"
targetNamespace="wlai/ApplicationView_SendDatagramMessage_request">
<xsd:element name="SendDatagramMessage">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="MessageDescriptor">
<xsd:complexType>
```

```
<xsd:sequence>
<xsd:element name="ExpirationPolicy" type="xsd:string" minOccurs="0"/>
<xsd:element name="Priority" type="xsd:string" minOccurs="0"/>
<xsd:element name="PersistPolicy" type="xsd:string" minOccurs="0"/>
<xsd:element name="CharacterSet" type="xsd:string" minOccurs="0"/>
<xsd:element name="Format" minOccurs="0">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="MQRFH2" minOccurs="0">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Encoding" type="xsd:long"/>
<xsd:element name="CodedCharSetId" type="xsd:long"/>
<xsd:element name="Format" type="xsd:string"/>
<xsd:element name="NameValueCCSID" type="xsd:long"/>
<xsd:element name="NameValueDatan" type="xsd:long"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="GroupMessageOptions" minOccurs="0">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="IsFirstMessage" type="xsd:boolean" minOccurs="0"/>
<xsd:element name="IsLastMessage" type="xsd:boolean" minOccurs="0"/>
```

```
<xsd:element name="IsIntermediateMessage" type="xsd:boolean"
minOccurs="0"/>
<xsd:element name="GroupId" type="xsd:string" minOccurs="0"/>
<xsd:element name="MsgSeqNumber" type="xsd:string" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="MessageId" type="xsd:string" minOccurs="0"/>
<xsd:element name="CorrelationId" type="xsd:string" minOccurs="0"/>
<xsd:element name="Data">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Format" type="xsd:string"/>
<xsd:element name="Content" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

SendMessage Response Schema

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="wlai/ApplicationView_SendDatagramMessage_response"
elementFormDefault="qualified"
targetNamespace="wlai/ApplicationView_SendDatagramMessage_response">
<xsd:element name="SendDatagramMessage">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Result">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="MessageId" type="xsd:string"/>
<xsd:element name="Status" type="xsd:string"/>
<xsd:element name="Error">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="CompletionCode" type="xsd:string" minOccurs="0"/>
<xsd:element name="ReasonCode" type="xsd:string" minOccurs="0"/>
<xsd:element name="ReasonCodeDescription" type="xsd:string"
minOccurs="0"/>
<xsd:element name="Trace" type="xsd:string" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="GroupMessage">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="GroupId" type="xsd:string" minOccurs="0"/>
<xsd:element name="MessageSeqNumber" type="xsd:string" minOccurs="0"/>
```

```
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

SendRequest Request Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="wlai/ApplicationView_SendRequestMessage_request"
elementFormDefault="qualified"
targetNamespace="wlai/ApplicationView_SendRequestMessage_request">
<xsd:element name="SendRequestMessage">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="MessageDescriptor">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="ExpirationPolicy" type="xsd:string" minOccurs="0"/>
<xsd:element name="Priority" type="xsd:string" minOccurs="0"/>
<xsd:element name="PersistPolicy" type="xsd:string" minOccurs="0"/>
<xsd:element name="CharacterSet" type="xsd:string" minOccurs="0"/>
<xsd:element name="Format" minOccurs="0">
<xsd:complexType>
```

```
<xsd:sequence>
<xsd:element name="MQRFH2" minOccurs="0">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Encoding" type="xsd:long"/>
<xsd:element name="CodedCharSetId" type="xsd:long"/>
<xsd:element name="Format" type="xsd:string"/>
<xsd:element name="NameValueCCSID" type="xsd:long"/>
<xsd:element name="NameValueDatan" type="xsd:long"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="GroupMessageOptions" minOccurs="0">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="IsFirstMessage" type="xsd:boolean" minOccurs="0"/>
<xsd:element name="IsLastMessage" type="xsd:boolean" minOccurs="0"/>
<xsd:element name="IsIntermediateMessage" type="xsd:boolean"
minOccurs="0"/>
<xsd:element name="GroupId" type="xsd:string" minOccurs="0"/>
<xsd:element name="MsgSeqNumber" type="xsd:string" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
```

```
</xsd:complexType>
</xsd:element>
<xsd:element name="Data">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Format" type="xsd:string"/>
<xsd:element name="Content" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

SendRequest Response Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="wlai/ApplicationView_SendRequestMessage_response"
elementFormDefault="qualified"
targetNamespace="wlai/ApplicationView_SendRequestMessage_response">
<xsd:element name="SendRequestMessage">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Result">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="MessageId" type="xsd:string"/>
<xsd:element name="Status" type="xsd:string"/>
```

```
<xsd:element name="Error">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="CompletionCode" type="xsd:string" minOccurs="0"/>
<xsd:element name="ReasonCode" type="xsd:string" minOccurs="0"/>
<xsd:element name="ReasonCodeDescription" type="xsd:string"
minOccurs="0"/>
<xsd:element name="Trace" type="xsd:string" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="GroupMessage">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="GroupId" type="xsd:string" minOccurs="0"/>
<xsd:element name="MessageSeqNumber" type="xsd:string" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

GetMessage Request Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="wlai/ApplicationView_GetTEXTMessage_request"
elementFormDefault="qualified"
targetNamespace="wlai/ApplicationView_GetTEXTMessage_request">

<xsd:element name="GetTEXTMessage">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="MessageId" type="xsd:string" minOccurs="0"/>

<xsd:element name="CorrelationId" type="xsd:string" minOccurs="0"/>

<xsd:element name="GroupId" type="xsd:string" minOccurs="0"/>

<xsd:element name="DataFormat" type="xsd:string"/>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:schema>
```

GetMessage Response Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="wlai/ApplicationView_GetTEXTMessage_response"
elementFormDefault="qualified"
targetNamespace="wlai/ApplicationView_GetTEXTMessage_response">

<xsd:element name="GetTEXTMessage">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Result">

<xsd:complexType>
```

```
<xsd:sequence>
<xsd:element name="Status" type="xsd:string"/>
<xsd:element name="Error">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="CompletionCode" type="xsd:string" minOccurs="0"/>
<xsd:element name="ReasonCode" type="xsd:string" minOccurs="0"/>
<xsd:element name="ReasonCodeDescription" type="xsd:string"
minOccurs="0"/>
<xsd:element name="Trace" type="xsd:string" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="GetInfo">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="QueueName" type="xsd:string"/>
<xsd:element name="MessageConsumption" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="PayLoad">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="MQMD">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="MessageType" type="xsd:string"/>
```

```
<xsd:element name="MessageId" type="xsd:normalizedString"/>
<xsd:element name="CorrelationId" type="xsd:normalizedString"/>
<xsd:element name="GroupId" type="xsd:normalizedString"/>
<xsd:element name="Format" type="xsd:normalizedString"/>
<xsd:element name="ReplyToQueueName" type="xsd:string"/>
<xsd:element name="ReplyToQueueManagerName" type="xsd:string"/>
<xsd:element name="UserId" type="xsd:string"/>
<xsd:element name="ApplicationIdData" type="xsd:string"/>
<xsd:element name="PutApplicationName" type="xsd:string"/>
<xsd:element name="PutDateTime" type="xsd:string"/>
<xsd:element name="ApplicationOriginData" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="Message">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="MQRFH2_Contents">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="StrucId" type="xsd:string" minOccurs="0"/>
<xsd:element name="Version" type="xsd:long" minOccurs="0"/>
<xsd:element name="StrucLength" type="xsd:long" minOccurs="0"/>
<xsd:element name="Encoding" type="xsd:long" minOccurs="0"/>
<xsd:element name="CodedCharSetId" type="xsd:long" minOccurs="0"/>
<xsd:element name="Format" type="xsd:string" minOccurs="0"/>
<xsd:element name="Flags" type="xsd:long" minOccurs="0"/>
```

```
<xsd:element name="NameValueCCSID" type="xsd:long" minOccurs="0"/>
<xsd:element name="NameValueLengthn" type="xsd:long" minOccurs="0"/>
<xsd:element name="NameValueDatan" type="xsd:long" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="Data">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Content" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Event Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="wlai/ApplicationView_EventforTEXT_event"
elementFormDefault="qualified"
targetNamespace="wlai/ApplicationView_EventforTEXT_event">
<xsd:element name="EventforTEXT">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="EventInfo">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="QueueName" type="xsd:string"/>
<xsd:element name="MessageConsumption" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="PayLoad">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="MQMD">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="MessageType" type="xsd:string"/>
<xsd:element name="MessageId" type="xsd:normalizedString"/>
<xsd:element name="CorrelationId" type="xsd:normalizedString"/>
<xsd:element name="GroupId" type="xsd:normalizedString"/>
<xsd:element name="Format" type="xsd:normalizedString"/>
```

```
<xsd:element name="ReplyToQueueName" type="xsd:string"/>
<xsd:element name="ReplyToQueueManagerName" type="xsd:string"/>
<xsd:element name="UserId" type="xsd:string"/>
<xsd:element name="ApplicationIdData" type="xsd:string"/>
<xsd:element name="PutApplicationName" type="xsd:string"/>
<xsd:element name="PutDateTime" type="xsd:string"/>
<xsd:element name="ApplicationOriginData" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="Message">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="MQRFH2_Contents">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="StrucId" type="xsd:string" minOccurs="0"/>
<xsd:element name="Version" type="xsd:long" minOccurs="0"/>
<xsd:element name="StrucLength" type="xsd:long" minOccurs="0"/>
<xsd:element name="Encoding" type="xsd:long" minOccurs="0"/>
<xsd:element name="CodedCharSetId" type="xsd:long" minOccurs="0"/>
<xsd:element name="Format" type="xsd:string" minOccurs="0"/>
<xsd:element name="Flags" type="xsd:long" minOccurs="0"/>
<xsd:element name="NameValueCCSID" type="xsd:long" minOccurs="0"/>
<xsd:element name="NameValueLengthn" type="xsd:long" minOccurs="0"/>
<xsd:element name="NameValueDatan" type="xsd:long" minOccurs="0"/>
</xsd:sequence>
```

```
</xsd:complexType>
</xsd:element>
<xsd:element name="Data" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Run-Time Parameter Values

This section lists the optional and required run-time parameter values that you can specify in a request document.

Table 2-1 Transaction Service Run-Time Parameter Values

Tag Name	Required	Permitted Values
TransactionFunction	Yes	Begin, Commit, BackOut

Table 2-2 SendMessage and SendRequest Service Run-Time Parameter Values

Tag Name	Required	Permitted Values
MessageDescriptor	Yes	
ExpirationPolicy	No	Any integer value (in tenths of a second)
Priority	No	Any value between 0 to 9 or AsQueueDef
PersistPolicy	No	Persistent, NotPersistent, AsQueueDef
CharacterSet	No	Refer toCharacter Set Catalog link on SendRequest and SendMessage pages during design-time
Format	No	None, String, MQRFH2

Table 2-2 SendMessage and SendRequest Service Run-Time Parameter Values (Continued)

MQRFH2	No Yes, if the format is MQRFH2	Encoding, CodedCharSetId, Format, NameValueCCSID, and NameValueDatan tags
GroupMessage Options	Yes, if Group Messaging is required	
IsFirstMessage	Yes, only for the first message in the group.	True
IsLastMessage	Yes, only for the last message.	True
IsIntermediateMessage	Yes, only for intermediate messages.	True
GroupId	No, for the first message. Yes, for intermediate and last messages.	A valid hexadecimal GroupId in string format
MsgSeqNumber	No, for the first message. Yes, for intermediate and last messages.	Positive integer
MessageId	No	A valid hexadecimal MessageId in string format
CorrelationId	Yes, for Reply message. No, for Datagram and Request Messages.	A valid hexadecimal CorrelationId in string format
Data	Yes	Format and Contents tag

Table 2-3 GetMessage Service Run-Time Parameter Values

Tag Name	Required	Permitted Values
MessageId	No	A valid hexadecimal MessageId in string format

Table 2-3 GetMessage Service Run-Time Parameter Values (Continued)

CorrelationId	No	A valid hexadecimal CorrelationId in string format
GroupId	No	A valid hexadecimal GroupId in string format
Data Format	Yes	TEXT, Binary, XML

APPENDIX C

Error Messages and Troubleshooting

This section lists error messages that you may encounter while using the Adapter for MQSeries. It describes what might have generated each error message, and what you can do to resolve the problem. It also contains troubleshooting tips on failures related to service execution.

Error Messages

The error messages along with suitable solutions to counter them are tabulated here:

Error	**javax.resource.spi.EISSystemException: An exceptional condition was encountered when <username> attempted to open a connection to the EIS; Message catalog not found**
Source	MQSeries Connection - Bindings
Description	This MQSeries error occurs when an invalid Queue Manager Name is entered.
Action	1. Check the Name of the Queue Manager. Verify it with the one that is available in the MQ Server. 2. Check whether the MQ Server is installed on the system on which the Adapter for MQSeries is installed. For details on configuring Connection Parameters, see "Configuring a Bindings Connection" on page 2-7.

Error	**javax.resource.spi.EISSystemException: An exceptional condition was encountered when <username> attempted to open a connection to the EIS; Message catalog not found**
Source	MQSeries Connection - TCP/IP
Description	This MQSeries error occurs when an invalid Queue Manager Name is entered.
Action	1. Check the Name of the Queue Manager. Verify it with the one that is available in the MQ Server. 2. Compare the Name of the Queue Manager Host, Queue Manager Channel and Queue Manager Port number with the ones that are available in the MQ Server. 3. Check if the CCSID is entered appropriately (check its presence in the CCSID Catalog). If it is entered, check the Language Support that is required for the CCSID entered and ensure that it is available on the system (the one you are trying to establish a connection with) on which the MQ Server is installed. For details on configuring Connection Parameters, see "Configuring a TCP/IP Connection" on page 2-8. Consult your MQ Server Administrator for more details.

Error	**javax.resource.spi.EISSystemException: An exceptional condition was encountered when <username> attempted to open a connection to the EIS; <name of the User Exit Class entered by the User>**
Source	MQSeries Connection - TCP/IP
Description	This MQSeries error occurs when you configure the User Exits incorrectly.
Action	Check whether one or more of the User Exits have been opted. If yes, check whether the App Name was entered for the displayed User Exit. If yes, check whether the Exit App Name Class Name is available in the execution environment. For details on using User Exits, see "Implementing User Exits" on page 2-8.

Error	Absence of an Event Response Document
Source	Events
Description	The expected Event Response Document is not generated after executing a workflow instance.
Action	1. Check whether the event is configured to the correct Queue. 2. Check for a message in the Queue. 3. Verify that the Connection parameters are appropriate. 4. Check whether the right data format is chosen. Also check whether the expected Message's data format matches the one configured to. 5. Check whether Content Filtering has been opted for. If yes, check whether the Content Filter Class name is correct and is available in the execution environment where the Adapter for MQSeries is deployed. 6. Check whether the Content Filter was developed correctly.

Troubleshooting Tips

These troubleshooting tips will be helpful in solving problems that you might encounter while executing the services.

- Check whether the service executed is a part of the Transaction Scope (if it is not configured as an implicit service).
- For explicit transaction services, check if the Transaction-Begin was invoked earlier. If not, check whether Transaction-Commit or BackOut was invoked before the Transaction-Begin.
- Ensure the Request Document matches the corresponding Request Schema.
- Find out if the mandatory tags and their values are provided in the Request Document.
- Check whether the Queue Name provided in the Service configuration is a valid one.
- Check the message holding capacity of the Queue to which the Service has been configured. This is applicable for both SendMessage and SendRequest Services.
- Check if the timeout period set in GetMessage is long enough for the Application to retrieve the message.
- Check whether the Queue in GetMessage contains the expected message.

- Confirm the presence of a CorrelationId in the Request Document of SendMessage Service with message type Reply.
- Check the validity of the MQRFH2 contents in confirmation with the MQRFH2 schema. This is applicable for both SendMessage and SendRequest Services.
- For XML payload (application data), check if the payload format provided in the request document is XML. The XML application data must be well-formed and Base64-encoded, and it must be placed within the Content tag of the request document.
- For Binary payload (application data), verify that the data format specified in the request document is Binary and or Base64-encoded.
- Check whether the GroupId and MsgSeqNumber (for Group Messaging) provided for the Last and Intermediate Messages in the Group are valid. This is applicable for SendMessage and SendRequest Services.
- Check if the Message User is authorized by the MQ Server. This is applicable when the Reports (COD, Exception, and Expiration) of SendMessage and SendRequest Services are not delivered to the specified Reply to Queue.

Index

F

G

I

L

M

P

R

S

T

U

W