BEA WebLogic Adapter for MQSeries

User Guide

Release 7.1
Document Date: June 2003

BEA WebLogic Adapter for MQSeries User Guide

Part Number	Date
N/A	June 2003

Contents

About This Document

This document explains how to use the BEA WebLogic Adapter for MQSeries to integrate IBM MQSeries messages with BEA WebLogic Integration. It is organized as follows:

- Chapter 1, “Introducing the BEA WebLogic Adapter for MQSeries,” introduces the BEA WebLogic Adapter for MQSeries and provides an overview of the adapter’s functionality.
- Chapter 2, “Defining Application Views for the Adapter for MQSeries,” describes how to define the application views for the Adapter for MQSeries.
- Chapter 3, “Using the Adapter for MQSeries,” describes in detail the services and events that the Adapter for MQSeries offers.
- Chapter 4, “Example of Using Services and Events,” illustrates examples of using the Adapter for MQSeries.
- Appendix A, “Schema Formats of Services and Events,” presents the formats for service and event schema created by the adapter.
- Appendix B, “Logging Messages,” provides information on logging levels and categories.
- Appendix C, “Run-Time Parameter Values,” details the run-time parameters used for adapter services.
- Appendix D, “Error Messages and Troubleshooting,” provides details on error messages and troubleshooting tips.

What You Need to Know

This document is written for system integrators who develop client interfaces between MQSeries and other Web applications. It describes how to use the BEA WebLogic Adapter for MQSeries to develop application environments with a special focus on message integration.

It is assumed that you know Web technologies and have a general understanding of MQSeries, Microsoft Windows, and UNIX systems.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the e-docs product documentation page at http://e-docs.bea.com.

Related Information

The following WebLogic Integration documents contain information that is relevant to using this product.

- *BEA WebLogic Adapter for MQSeries Installation and Configuration Guide* at http://edocs.bea.com/wladapters/mq/docs71/index.html
- *BEA WebLogic Adapter for MQSeries Release Notes* at http://edocs.bea.com/wladapters/mq/docs71/index.html
- WebLogic Server installation and user documentation at http://edocs.bea.com/more_wls.html
- BEA WebLogic Integration installation and user documentation at http://edocs.bea.com/more_wli.html

This document assumes that you have in-depth knowledge of Workflow Design, Workflow Templates, Worklists, and WebLogic Integration Studio.

If you do not have the required knowledge of workflows or the WebLogic Integration Studio, see the following documents:

- *Using the WebLogic Integration Studio* at http://edocs.bea.com/wli/docs70/studio/index.htm
- *Learning to Use BPM with WebLogic Integration* at http://edocs.bea.com/wli/docs70/bpmtutor/index.htm

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 the BEA WebLogic Adapter for MQSeries 7.1 release.

If you have any questions about this version of BEA WebLogic Adapter for MQSeries, or if you have problems installing and running the product, 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.

`monospace text`	Indicates code samples, commands, and their options, data structures and their members, data types, directories, and filenames and their extensions.
	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. *Examples*: LPT1 SIGNON OR

Convention	Item
`{ }`	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.

`...`	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 explains the key concepts. It contains the following topics:

- About the BEA WebLogic Adapter for MQSeries
- Key Concepts

About the BEA WebLogic Adapter for MQSeries

The BEA WebLogic Adapter for MQSeries integrates your IBM MQSeries messages with WebLogic Integration. You can use the adapter to exchange XML, Binary, and TEXT formats between your MQSeries resources and the WebLogic Integration.

Supported Integration Capabilities

MQSeries messaging products support application integration by sending and receiving data as messages that allow business applications to exchange information across platforms. They account for network interfaces, delivery of messages, deal with communication protocols, dynamically distribute workload across available resources, handle recovery after system problems, and help make programs portable. This allows programmers to use their skills to handle key business requirements, instead of wrestling with underlying network complexities.

The BEA WebLogic Adapter for MQSeries provides:

- Asynchronous, bi-directional message interactions between WebLogic Integration and native IBM 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.

Key Concepts

This topic describes the following concepts, which you must understand before using the BEA WebLogic Adapter for MQSeries:

- Application Views
- Services
- Events
- Schemas

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. Application views define:

- 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 the 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.

An application view is typically configured for a single business purpose and contains only the services or events required for that business purpose.

Application views provide a layer of abstraction between applications and the EIS, making it easier for developers and non-programmers to access and maintain the services and events exposed by the adapter.

Services

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. Responses can be either synchronous or asynchronous. When an application receives a request to invoke a business service, the application view invokes the service in the target application and responds with an XML document that describes the results.

To define a service, you must define input requirements, output expectations, and an interaction specification containing static secondary metadata about the request.

A service receives an `XML` request document from a client and invokes the associated function in the underlying EIS. Services are consumers of messages. They may or may not provide responses. A service may be invoked in either of two ways: synchronously or asynchronously, in a workflow. When a synchronous service is used, the client waits for the response before proceeding with processing. When an asynchronous service is used, the client application issues a service request and then proceeds with processing without waiting for the response.

A service performs the following functions:

- Receive service requests from an external client.
- Transform an `XML` request document into the EIS-specific format. The request document conforms to the request `XML` schema for the service.
- Invoke the underlying function in the EIS and wait for a response from that function.
- Transform the response from the EIS-specific data format to an `XML` format that conforms to the response `XML` schema for the service.

Events

Events are asynchronous, one-way messages received from an EIS. For example, the adapter can receive a message from an MQ system. The adapter routes the EIS message to the appropriate software component.

Events are triggers to workflows. When a particular message arrives in a queue, it triggers an event to read the message, and convert it, if necessary, to an XML format that matches the required schema. The event then initiates a workflow.

An event is an `XML` document published by an application view when an occurrence of interest takes place within an EIS. Clients that want to be notified of events, request such notification by registering with an application view. The application view then acts as a broker between the target application and the client. When a client has subscribed to events published by an application view, the application view notifies the client whenever an event of interest occurs in the target application. Upon receiving such notification, the event is passed as an `XML` document that describes the event. Application views that publish events can also provide clients with the `XML` schema for publishable events.

Events are designed to propagate information from an EIS to WebLogic Server. They can be described as publishers of information.

Events running in a WebLogic Integration environment perform the following functions:

- Respond to events that occur inside the running EIS by extracting and storing data about the event from the EIS.
- Transform event data from an EIS-specific format to an `XML` document that conforms to the `XML` schema for the event.
- Propagate each event to an event context obtained from the application view by using the event router.

Schemas

At run-time, the EIS and the adapter exchange service requests, service responses, and events via XML documents. The adapter handles the data translation between XML documents and the EIS format, using schemas that have been defined at design-time to map the data between XML and the EIS format:

- For service requests, the request arrives at the adapter in the form of an XML document. The adapter uses the request schema associated with the service (as defined in the application view) to translate the request to the format that the EIS expects. Similarly, when the adapter receives the response back from the EIS, it uses the response schema associated with the service to translate the response to an XML document that the requesting application handles.
- For event notifications, the inbound message arrives at the adapter in the format that the EIS uses to publish the event. The adapter uses the event schema associated with the event (as defined in the application view) to translate the response to an XML document that the subscribed application handles.

The Adapter for MQSeries creates the schema you need when you define services and events. To view the schemas that the adapter creates, see Appendix A, “Schema Formats of Services and Events.”

2 Defining Application Views for the Adapter for MQSeries

This section explains how to define application views for the Adapter for MQSeries. It contains the following topics:

- Before You Begin
- Defining and Deploying an Application View
- Editing an Application View

Before You Begin

When you define an application view, you create an XML-based interface between WebLogic Server and the Adapter for MQSeries. Once you create the application view, a business analyst can use it to create business processes that use the application. You can create as many application views for an adapter, each of which may contain any number of services and events.

Note: Before you define an application view, determine which business processes should be supported by the application view that you are configuring. 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.

Defining and Deploying an Application View

This section explains how to define and deploy application views using an Adapter for MQSeries.

This procedure consists of the following steps:

- Step 1. Log On to the Application View Console
- Step 2. Add a Folder
- Step 3. Define an Application View
- Step 4. Establish an MQSeries Connection
- Step 5. Add Services and Events
- Step 6. Deploy the Application View
- Step 7. Test Services and Events

Note: Before performing the following steps, ensure that the WebLogic Integration Server is running on your system.

Step 1. Log On to the Application View Console

The Application View Console displays all the application views in your WebLogic Integration environment, organized in folders.

To log on to the Application View Console:

1. Open a new browser window.

2. Enter the URL for your system's Application View Console. The actual URL you enter depends on your system. It should conform to the following format:

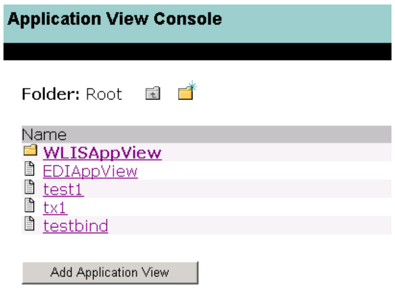
   `http://`*`localhost`*`:`*`port`*`/wlai`

   Here, *`localhost`* represents the IP address machine on which the WebLogic Integration Server is running and *`port`* represents the listening port.

   The Application View Console Logon page appears.

3. Enter your WebLogic Server username and password, and click Login. The Application View Console page appears.

Figure 2-1 Application View Console



Step 2. Add a Folder

You organize the application views in folders. A single folder may contain both applications views and other subfolders. Once you create a folder, you cannot move it to another folder. Before you can remove a folder, you must first remove all applications views and subfolders. Once you create an application view in a folder, you can remove the application view, but you cannot move it to another folder. You can add application views to existing folders.

To add a folder:

1. On the Application View Console page, click the New Folder icon. The Add Folder page appears.

2. Enter a name in the New Folder field. Any valid Java Identifier is allowed to be the name.

3. Click Save. The newly created folder appears on the Application View Console page.

Step 3. Define an Application View

You must define an application view and relate it to the adapter you are working with. For more information about defining application views and using them in workflows, see *Using Application Integration*:

- For WebLogic Integration 7.0, see `http://edocs.bea.com/wli/docs70/aiuser/2usrdef.htm`
- For WebLogic Integration 2.1, see `http://edocs.bea.com/wlintegration/v2_1sp/aiuser/2usrdef.htm`

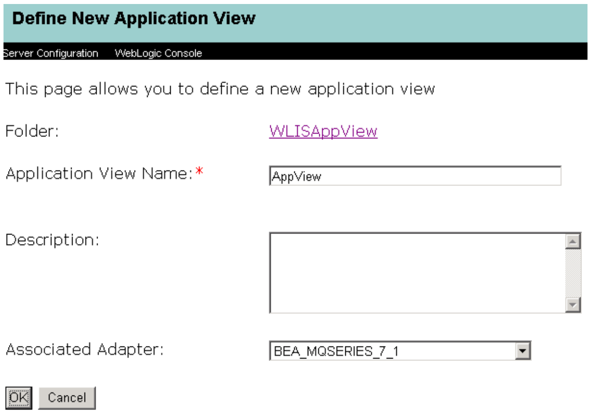
To define an application view:

1. Click the newly created folder. It takes you to the page where you can add application views to this folder.
2. Click Add Application View.

   Note: Make sure you are working in the appropriate folder before performing this step. Once you define an application view, you cannot move it to another folder.

   The Define New Application View page appears.

Figure 2-2 Define New Application View



This page allows you to define a new application view

Folder: WLISAppView

Application View Name:* AppView

Description:

Associated Adapter: BEA_MQSERIES_7_1

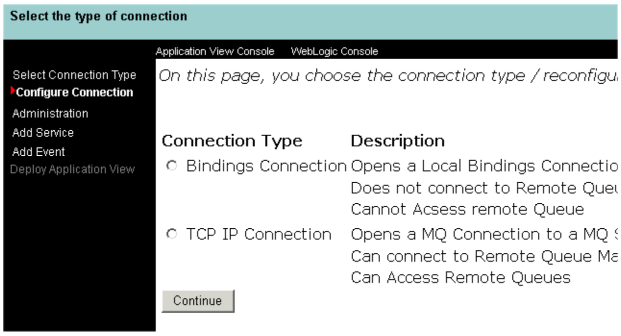
Note: A red asterisk (*) indicates that a field is mandatory.

3. Enter a name for the application view. Generally, the name should indicate the functions performed by this application. Each application view must be unique to its Adapter. Example, `my_mqseries_appview`.
4. Select an adapter. Example, `BEA_MQSERIES_7_1`.
5. Click OK. The Select the Type of Connection page appears.

Step 4. Establish an MQSeries Connection

You can connect to MQSeries either by using a bindings connection or a TCP/IP connection.You can choose the connection type on the Select the Type of Connection page.

Figure 2-3 Select the Type of Connection Page


Select the type of connection

Application View Console WebLogic Console

Select Connection Type
Configure Connection
Administration
Add Service
Add Event
Deploy Application View

On this page, you choose the connection type / reconfigu

Connection Type	Description
○ Bindings Connection	Opens a Local Bindings Connectio Does not connect to Remote Queu Cannot Acsess remote Queue
○ TCP IP Connection	Opens a MQ Connection to a MQ Can connect to Remote Queue Ma Can Access Remote Queues

Continue

You can choose one of the following types of MQ Connection:

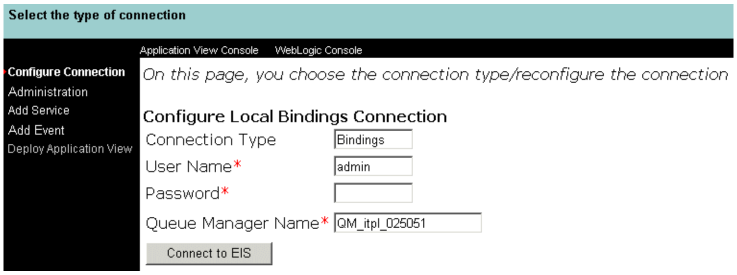
- Bindings Connection: provides an MQ Connection to the MQ Server running on the system where the Adapter for MQSeries is installed. To learn more, see Step 4A. Connecting Through Bindings Connection Type.
- TCP/IP Connection: provides the option of connecting to any MQ Server running on the same network where the Adapter for MQSeries is installed. To learn more, see Step 4B. Connecting Through TCP/IP Connection Type.

Step 4A. Connecting Through Bindings Connection Type

To establish a connection to MQSeries through bindings, do the following:

1. On the Select Type of Connection page, select Bindings Connection. The Configure Local Bindings Connection page appears.

Figure 2-4 Configure Local Bindings Connection


Select the type of connection

Application View Console WebLogic Console

Configure Connection
Administration
Add Service
Add Event
Deploy Application View

On this page, you choose the connection type/reconfigure the connection

Configure Local Bindings Connection

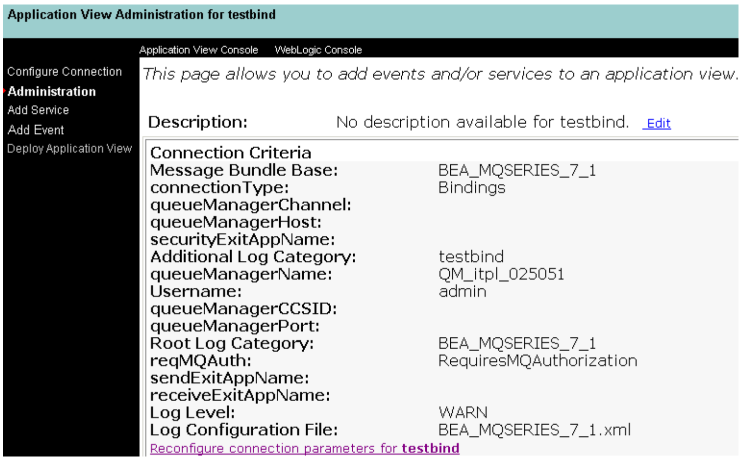
Connection Type	Bindings
User Name*	admin
Password*	
Queue Manager Name*	QM_itpl_025051

Connect to EIS

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

2. Enter the WebLogic User Name and Password.
3. Enter the name of the Queue Manager.
4. Click Connect to EIS. The Application View Administration page appears with the summary of the connection.

Figure 2-5 Application View Administration - Local Bindings Connection Summary



This page allows you to add events and/or services to an application view.

Description: No description available for testbind. Edit

Connection Criteria

Message Bundle Base:	BEA_MQSERIES_7_1
connectionType:	Bindings
queueManagerChannel:	
queueManagerHost:	
securityExitAppName:	
Additional Log Category:	testbind
queueManagerName:	QM_itpl_025051
Username:	admin
queueManagerCCSID:	
queueManagerPort:	
Root Log Category:	BEA_MQSERIES_7_1
reqMQAuth:	RequiresMQAuthorization
sendExitAppName:	
receiveExitAppName:	
Log Level:	WARN
Log Configuration File:	BEA_MQSERIES_7_1.xml

Reconfigure connection parameters for **testbind**

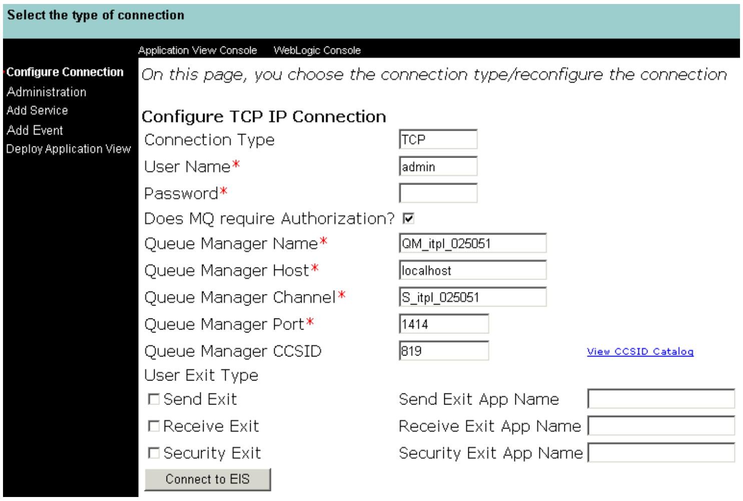
Step 4B. Connecting Through TCP/IP Connection Type

To learn more about the options available when you select a TCP/IP connection, see “About TCP/IP Connections” on page 3-1.

To establish a connection to through TCP/IP:

1. On the Select Type of Connection page, select TCP/IP Connection. The Configure TCP/IP Connection page appears.

Figure 2-6 Configure TCP IP Connection



Select the type of connection

On this page, you choose the connection type/reconfigure the connection

Configure TCP IP Connection

Connection Type	TCP
User Name*	admin
Password*	
Does MQ require Authorization?	✓
Queue Manager Name*	QM_itpl_025051
Queue Manager Host*	localhost
Queue Manager Channel*	S_itpl_025051
Queue Manager Port*	1414
Queue Manager CCSID	819

User Exit Type

☐ Send Exit	Send Exit App Name	
☐ Receive Exit	Receive Exit App Name	
☐ Security Exit	Security Exit App Name	

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

2. Enter the WebLogic User Name and Password.
3. Select the check box if MQSeries requires an authorization.

   Note: If you select this, the User Name and Password used here should be valid on both the WebLogic Platform and the MQ Server.

4. Enter the name of the Queue Manager.
5. Enter the name of the host where the MQ Server is installed. Example, `localhost`.
6. Enter the name of the channel to connect to the Queue Manager.
7. Enter the port number of the Queue Manager. Example, `1414`.
8. If you expect to exchange messages using a non-latin character set, enter the character set ID from the CCSID Catalog.

   CCSID defines the character set of character data in the message. Click the View CCSID Catalog link for a list of character catalogs. To learn more, see “Using CCSID in MQ Connection” on page 3-3.

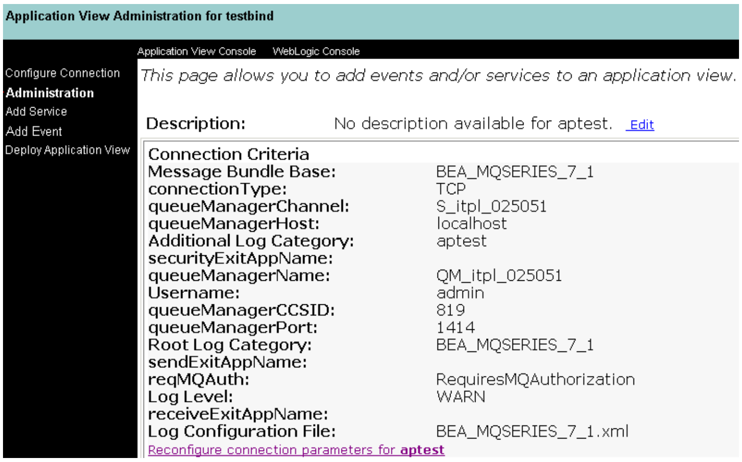
9. Select the applicable User Exit Type and enter the relevant application name. These are the user exits:

- Send Exit: processes the message before sending it to a queue.
- Receive Exit: processes the message after getting it from a queue.
- Security Exit: performs a security check on the message.

  To learn more about user exit types, see "Implementing User Exits" on page 3-2.

10. Click Connect to EIS. The Application View Administration page appears with the summary of the connection.

Figure 2-7 Application View Administration - TCP/IP Connection Summary



This page allows you to add events and/or services to an application view.

Description: No description available for aptest. Edit

Connection Criteria	
Message Bundle Base:	BEA_MQSERIES_7_1
connectionType:	TCP
queueManagerChannel:	S_itpl_025051
queueManagerHost:	localhost
Additional Log Category:	aptest
securityExitAppName:	
queueManagerName:	QM_itpl_025051
Username:	admin
queueManagerCCSID:	819
queueManagerPort:	1414
Root Log Category:	BEA_MQSERIES_7_1
sendExitAppName:	
reqMQAuth:	RequiresMQAuthorization
Log Level:	WARN
receiveExitAppName:	
Log Configuration File:	BEA_MQSERIES_7_1.xml

Reconfigure connection parameters for **aptest**

Step 5. Add Services and Events

You can add services and events that support specific business processes. An application view can have multiple services and events. The required business processes determine the types of services and events you include in your application view.

Step 5A. Add a Service to an Application View

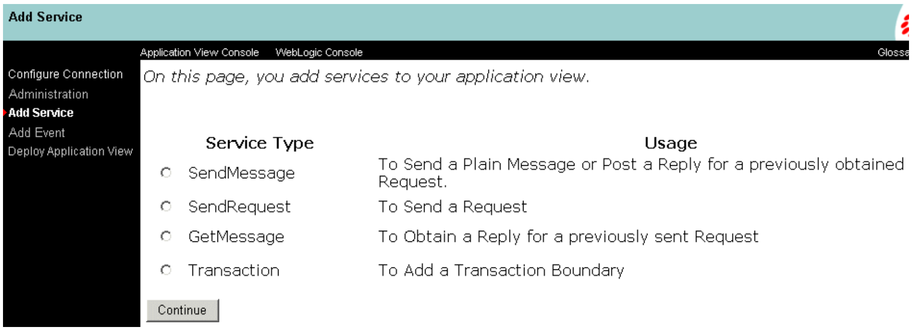
You must consider the following points before you execute a workflow using Adapter for MQSeries services:

- The SendMessage, SendRequest, and GetMessage services must always be a part of an existing transaction scope.
- Within a transaction scope, the SendMessage, SendRequest, and GetMessage services can be invoked in any order.
- Transaction boundary services (Begin and Commit/BackOut) are required for sending and receiving messages.

To add a service:

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

Figure 2-8 Add Service



On this page, you add services to your application view.

Service Type	Usage
○ SendMessage	To Send a Plain Message or Post a Reply for a previously obtained Request.
○ SendRequest	To Send a Request
○ GetMessage	To Obtain a Reply for a previously sent Request
○ Transaction	To Add a Transaction Boundary

2. Select the Service Type and click Continue. A page for the service type you select appears. The services provided by the Adapter for MQSeries are as follows:
   - Transaction Services: begins and ends a transaction that can contain any number of other services. To learn more about Transaction services, see “About Transaction Services” on page 3-4.
   - SendMessage: sends a message to a specified queue.
   - SendRequest: sends a request message to a specific queue.
   - GetMessage: gets a message from a specified queue.

     For specific instructions on configuring a GetMessage service, see “About Configuring GetMessage Services” on page 2-13.
3. Enter a unique name for the service.

4. Select the message type from the drop-down list. The options are:
   - Datagram: for your messages where no reply is expected. This message type is used for SendMessage services. For more information, see "Sending a Datagram Message" on page 3-7.
   - Reply: for your reply messages. This message type is used for SendMessage services. For more information, see "Sending a Reply Message" on page 3-8.
   - Request: for your message where a reply is expected. This message type is used for SendRequest services.
5. Enter a valid queue name for the queue manager you are connecting to, for Bindings connection. For a TCP/IP connection enter a valid queue name and sender channel for the queue manager. For details on setting up a sender-receiver channel to the remote queue manager, see "Sending Messages to Remote Queues" on page 3-9.

Note: The instructions that follow apply to SendMessage and SendRequest services. For instructions for configuring GetMessage services, see "About Configuring GetMessage Services" on page 2-13.

6. Enter the expiration value for the message.
7. Select the message priority. You can select a number between 0 and 9, or AsQueuedef (as defined in the queue).
8. Select the persistence policy. The available options are Persistent, NotPersistent, and AsQueuedef.
9. If you expect to exchange messages using a non-latin character set, enter the character set ID from the CCSID Catalog.

   CCSID defines the character set of character data in the message. Click the View CCSID Catalog link for a list of character catalogs. To learn more, see "Using CCSID in MQ Connection" on page 3-3.
10. Enter a message user name that is authorized by the MQSeries Server and a member of the MQ Administrator group.
11. Select a segmentation policy. If you select Allowed, the MQ Manager can segment the message, if required. If you select NotAllowed, the message cannot be segmented.
12. Select the relevant report messaging option:

- COA: Confirmation of Arrival, with some data or with full data.
- COD: Confirmation of Delivery, with some data or with full data.
- Exception report, with some data or with full data.
- Expiration report, with some data or with full data.

13. If you select a report messaging option, enter the name of the queue where the report should be sent. The default value is obtained from the queue manager to which the connection is made.

    Note: The reports go to the specified reply to queue if the user is authorized by MQSeries. Otherwise, they go into the dead letter queue of the queue manager where the message is sent. In any case, the COA report goes into the specified Reply to Queue Name destination. For more information on reports, see your MQSeries documentation.

14. Enter the queue manager to which the report should be sent. The default value is obtained from the queue manager to which the connection is made. This is an optional field.

15. Select the applicable format for the message. The available options are None, String, and MQRFH2. To view the format of MQRFH2, click View Format link.

    Note: If you do not specify a value in run-time, the design-time value takes precedence. If you specify both run-time and design-time values, the run-time value takes precedence.

16. Enter the contents of the selected format. This field is mandatory if you select MQRFH2 data format. To create the contents, use the MQRFH2 schema and insert values in it. For more information, see Sending Messages with MQRFH2 Header Information.

17. Select the data format. If you specify the data format as TEXT or Binary during design-time, you cannot specify the data format as XML during run-time.

18. Enter the schema of the XML data content. This is required only when you select XML as the data format.

    Note: The XML data content schema cannot be provided during run-time.

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

About Configuring GetMessage Services

When you configure a GetMessage service, you specify the service name, service type, description, and queue name the same as for a SendMessage or SendRequest service.

In addition, you specify a message consumption setting and a time out.

The Message Consumptions options are:

- Browse – This option is only for Browsing a message. Even after the message is read from the Queue it remains there intact.
- Delete – This option is to fully consume the message as it is read. After the message is read, it is permanently removed from the Queue.

Note: It is the MQ Administrator's responsibility to periodically remove outdated and unwanted messages from the Queue whenever the Browse is set as the consumption type.

The Time Out is the maximum amount of time for which the application should pause before getting a message. For unlimited waiting period, specify -1.

When you configure a GetMessage service for TEXT or Binary Data formats, it will not fetch messages of XML format. The application will throw a Resource Exception on such occurrences. When a GetMessage service is configured for XML Data formats it will not fetch messages of TEXT or Binary formats. The application will throw a Resource Exception on such occurrences. At run-time, one or more or all of the following values can be provided.

- MessageId: The MessageId of the message that is to be received.
- CorrelationId: The CorrelationId of the message that is to be received.
- GroupId: The GroupId of the message that is to be received. For more information, see "Sending Group Messages" on page 3-9.

A Message that matches these given IDs is received. If none of the above is provided, the first message in the Queue is received.

Messages, also known as In-process replies, may be responses to previously sent requests and/or subsequent messages in a group. For more information on the data formats, see "Using Data Formats in Services and Events" on page 3-14.

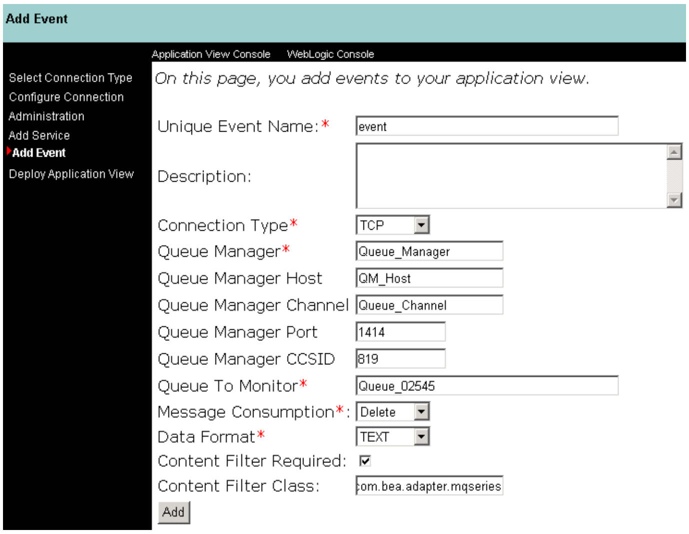
Step 5B. Add an Event to an Application View

You can use an Adapter for MQSeries event to monitor a queue for messages. Messages can be received only from queues that are local to the queue manager to which the connection is obtained and not from remote queues.

To add an event:

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

Figure 2-9 Add Event



On this page, you add events to your application view.

Unique Event Name:*	event
Description:	
Connection Type*	TCP
Queue Manager*	Queue_Manager
Queue Manager Host	QM_Host
Queue Manager Channel	Queue_Channel
Queue Manager Port	1414
Queue Manager CCSID	819
Queue To Monitor*	Queue_02545
Message Consumption*:	Delete
Data Format*	TEXT
Content Filter Required:	✓
Content Filter Class:	:om.bea.adapter.mqseries

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

2. Enter a Unique Name for the event.
3. Select the type of connection from the drop-down list.
4. Enter the name of the Queue Manager.
5. Enter the name of the host required to connect to the Queue Manager.

   Note: The host name should be a valid IP address on a non-Windows platform. Example, `172.19.138.44`. On Windows, it can be either an IP address or a valid name. Example, `QM_Host`

6. Enter the name of the Queue Manager Channel. This entry is required only if the connection type is TCP/IP.
7. Enter the port number of the Queue Manager. This entry is required only when the connection type is TCP/IP.
8. Enter the name of the queue to monitor.
9. If you expect to exchange messages using a non-latin character set, enter the character set ID from the CCSID Catalog.

   CCSID defines the character set of character data in the message. Click the View CCSID Catalog link for a list of character catalogs. To learn more, see “Using CCSID in MQ Connection” on page 3-3.
10. Select the type of message consumption from the drop-down list.
    - Browse – This option is only for Browsing a message. Even after the message is read from the Queue it remains there intact.
    - Delete – This option is to fully consume the message as it is read. After the message is read, it is permanently removed from the Queue.

    Note: It is the MQ Administrator’s responsibility to periodically remove outdated and unwanted messages from the Queue whenever the Browse is set as the consumption type.
11. Select the data format of the message from the drop-down list. For details, see “Using Data Formats in Services and Events” on page 3-14
12. Select the Content Filter Required check box if you need content filtering. For details, see “About Content Filtering in Events” on page 3-5.
13. Enter the class of the content filter. You should enter this only if you select the Content Filter Required check box.
14. Click Add. The event is added to the application view.

Step 6. Deploy the Application View

After adding the services and events, you must deploy the application view to use its services and events in a workflow. If you want to reconfigure the application view, you have to undeploy it and reconfigure. For details on reconfiguring an application view, see “Editing an Application View” on page 2-30.

To deploy the application view, do the following:

1. After you add services and events, on the Application View Administration page, click Continue. The Deploy Application View page appears.

   Note: Before you deploy the application view, configure the log verbosity level for the application view. For more details, see Appendix B, “Logging Messages.”

2. Click Deploy. The application view is deployed.

Step 7. Test Services and Events

The purpose of testing an application view service is to evaluate whether or not that service interacts properly with the MQSeries. After deploying an application view, you can test the services and events in one of the following ways:

- Using the Application View Console
- Using WebLogic Integration Studio

Note: Before you test an application view, ensure that all the required services and events are added to it and it has been deployed.

Step 7A. Test Services and Events Using the Application View Console

It's a good idea to test the functionality of an application view before you start using it in workflow. You can do this using the Application View Console.

Note: Before you test an application view’s services, ensure that the application view contains a Transaction service and other necessary services that form a part of the transaction scope.

To test a service, see “Test Services in Application View Console” on page 2-17.

To test an event, see “Test Events in the Application View Console” on page 2-19.

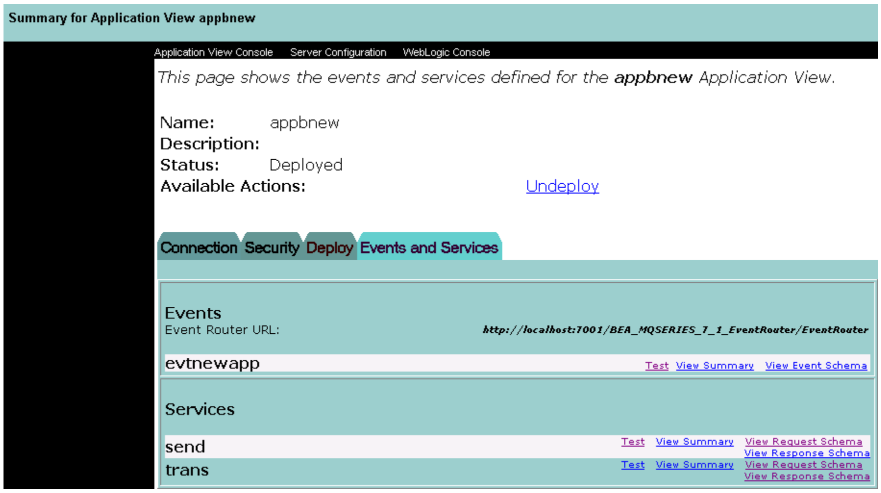
Test Services in Application View Console

To confirm that a deployed application view’s services are correctly configured, you must test it before using them in the workflow. You can test the SendMessage, SendRequest, and GetMessage services.

To test an application view’s services, do the following:

1. Open the WebLogic Integration-Application View Console, and select a deployed application view. The Summary for Application View page appears.

Figure 2-10 Summary for Application View



This page shows the events and services defined for the ***appbnew*** *Application View.*

Name: appbnew
Description:
Status: Deployed
Available Actions:

Events
Event Router URL: http://localhost:7001/BEA_MQSERIES_7_1_EventRouter/EventRouter

evtnewapp

Services

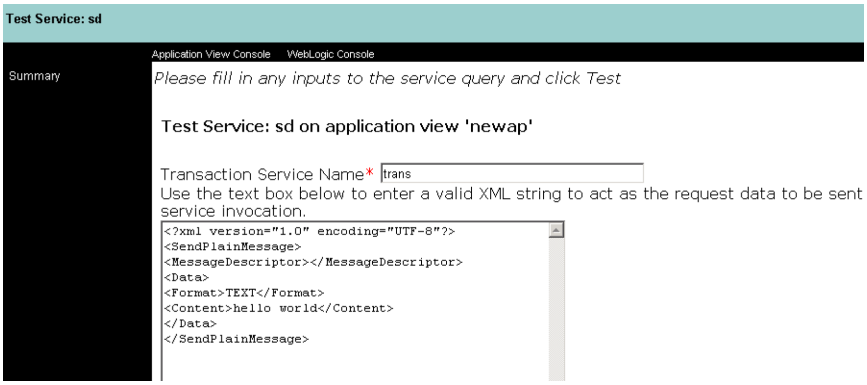
send

trans

2. Click the Test link for one of the application view’s services (SendMessage, SendRequest, and GetMessage).

   The Test Service page for that service appears.

Figure 2-11 Test Service


Test Service: sd

Application View Console   WebLogic Console

Summary

Please fill in any inputs to the service query and click Test

Test Service: sd on application view 'newap'

Transaction Service Name* trans

Use the text box below to enter a valid XML string to act as the request data to be sent service invocation.

```
<?xml version="1.0" encoding="UTF-8"?>
<SendPlainMessage>
<MessageDescriptor></MessageDescriptor>
<Data>
<Format>TEXT</Format>
<Content>hello world</Content>
</Data>
</SendPlainMessage>
```

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

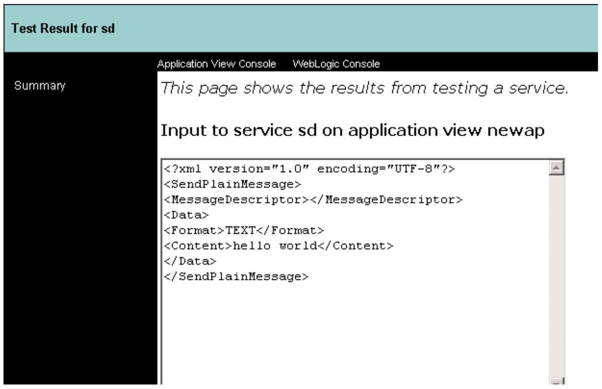
3. Enter the name of the transaction service (that has been added to the selected application view) in the Transaction Service Name field.

4. Enter a sample request document that matches the request schema for the service to be tested, in the text box. For example,

```
<?xml version="1.0" encoding="UTF-8"?>
<SendPlainMessage>
<MessageDescriptor></MessageDescriptor>
<Data>
<Format>TEXT</Format>
<Content>hello world</Content>
</Data>
</SendPlainMessage>
```

5. Click Test to execute the service. The Test Result page appears.

Figure 2-12 Test Result



This page shows the results from testing a service.

Input to service sd on application view newap

```
<?xml version="1.0" encoding="UTF-8"?>
<SendPlainMessage>
<MessageDescriptor></MessageDescriptor>
<Data>
<Format>TEXT</Format>
<Content>hello world</Content>
</Data>
</SendPlainMessage>
```

You have now confirmed that the application view service is correctly configured.

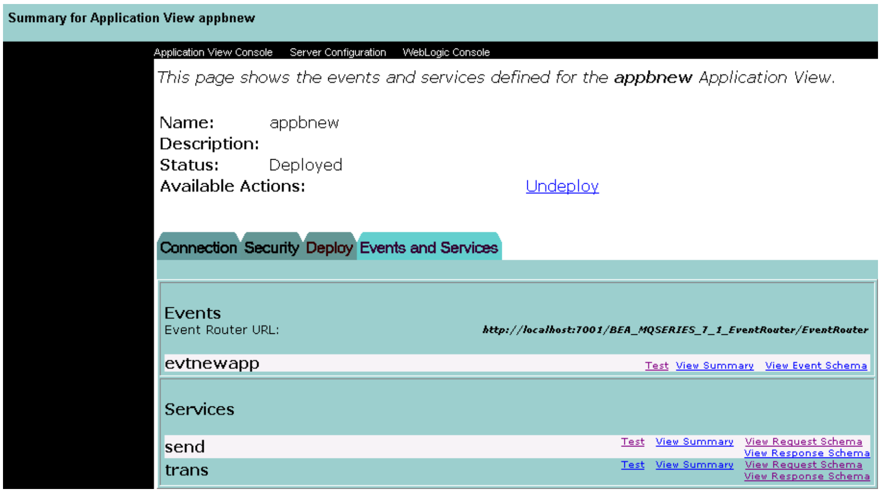
Test Events in the Application View Console

To confirm that a deployed application view's events are correctly configured, you must test it before using them in the workflow. You can test an event through a service or manually. Each of these is described in the following steps.

To test an event, do the following:

1. Open the WebLogic Integration-Application View Console, and click the application view bearing deployed status.The Summary for Application View page appears.

Figure 2-13 Summary for Application View



This page shows the events and services defined for the ***appbnew*** *Application View.*

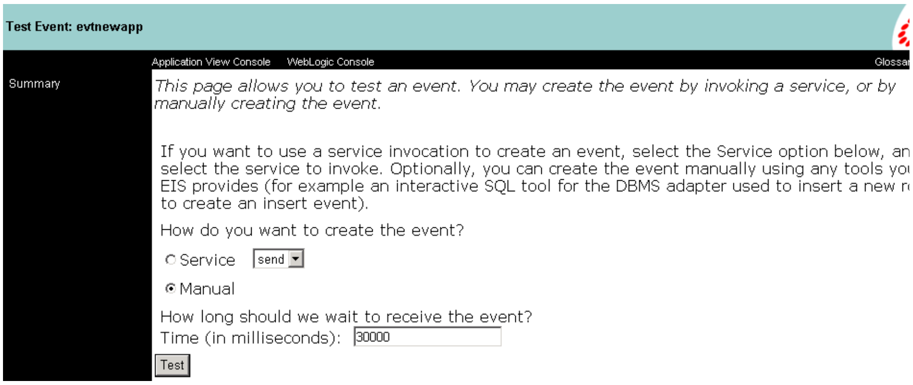
Name: appbnew
Description:
Status: Deployed
Available Actions: Undeploy

Events
Event Router URL: *http://localhost:7001/BEA_MQSERIES_7_1_EventRouter/EventRouter*
evtnewapp

Services
send
trans

2. Click the Test link for one of the application view's events. The Test Event page appears.

Figure 2-14 Test Event



This page allows you to test an event. You may create the event by invoking a service, or by manually creating the event.

If you want to use a service invocation to create an event, select the Service option below, an
select the service to invoke. Optionally, you can create the event manually using any tools yo
EIS provides (for example an interactive SQL tool for the DBMS adapter used to insert a new r
to create an insert event).

How do you want to create the event?

Service [send]
[✓ Manual]

How long should we wait to receive the event?
Time (in milliseconds): 30000

3. To test the event through a service, do the following:

   a. Select Service, and select the service from the drop-down list. The Test Service page appears as shown in the figure "Test Service" on page 2-18.

   b. Enter the name of the transaction service (that has been added to the selected application view) in the Transaction Service Name field.

   c. Enter a sample Request Document that matches the Request Schema for the selected service, in the text area.

   d. Click Test to test the event using a service. The Test Result page appears.

4. To test the event directly, do the following

   a. Select Manual and enter the waiting period in the Time (in milliseconds) in the field.

   b. Click Test to test the event. The Test Result page appears.

   Note: If the duration of waiting is lesser than the actual time required to pick the message from the Queue, the Result page will display the Timed Out message. In such case, you will have to increase the waiting period specified in the Time field.

   You have now confirmed that the application view is correctly configured and can receive events.

   Note: If you wait longer than the specified period and do not receive the event's result, you should assume that there is a problem with the application view event. Examine the WebLogic Server log for information about the event's activity.

Step 7B. Test Services and Events Using WebLogic Integration Studio

You can test events and services separately, using the WebLogic Integration Studio. Before you begin testing services and events, you must complete the tasks described in the following steps.

Note: The information provided here assumes that you have in-depth knowledge of Workflow Design, Workflow Templates, Worklists, and WebLogic Integration Studio.

If you do not have the required knowledge of workflows or the WebLogic Integration Studio, see the following documents:

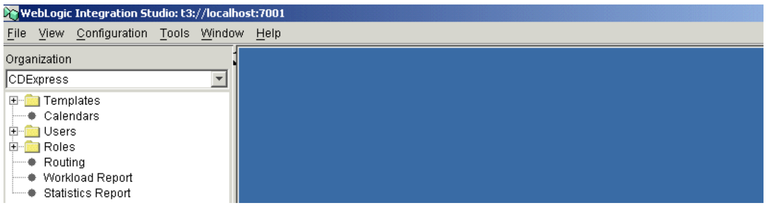
- *Using the WebLogic Integration Studio* at http://edocs.bea.com/wli/docs70/studio/index.htm
- *Learning to Use BPM with WebLogic Integration* at http://edocs.bea.com/wli/docs70/bpmtutor/index.htm

Before you test an application views's services and events, do the following:

1. Create the necessary Request Documents for the services that you want to test. You can create the Request Documents using the Request Schema available for that service.

2. Log on to the WebLogic Integration Studio, using a valid user name, password, and server URL. The WebLogic Integration Studio appears.

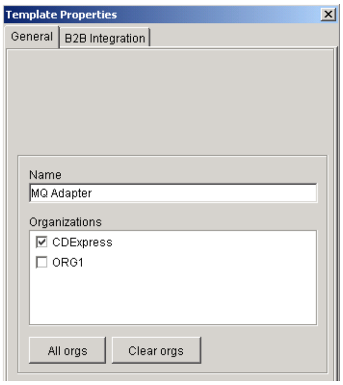
Figure 2-15 WebLogic Integration Studio


WebLogic Integration Studio: t3://localhost:7001
File View Configuration Tools Window Help
Organization
CDExpress
Templates
Calendars
Users
Roles
Routing
Workload Report
Statistics Report

3. In the left pane of WebLogic Integration Studio, from the Organizaton drop-down list, select the relevant organization.

4. In the left pane, right-click the Templates folder and select Create Template. The Template Properties dialog box appears.

Figure 2-16 Template Properties Dialog Box



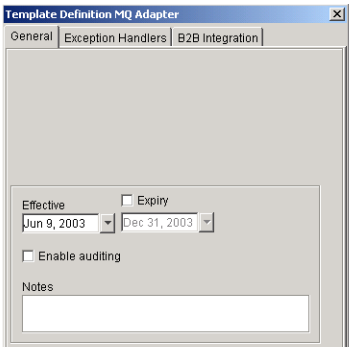
Name: MQ Adapter

☑ CDExpress
☐ ORG1

5. On the General tab, in the Name field, enter a name for the template, and click OK. The new template (MQ Adapter) appears in the Templates folder, on the left pane of the WebLogic Integration Studio.

6. Right-click the newly created template, and select Create Template Definition. The Template Definition MQ Adapter dialog box appears.

Figure 2-17 Template Definition Dialog Box



Template Definition MQ Adapter

General | Exception Handlers | B2B Integration

Effective: Jun 9, 2003

☐ Expiry: Dec 31, 2003

☐ Enable auditing

Notes

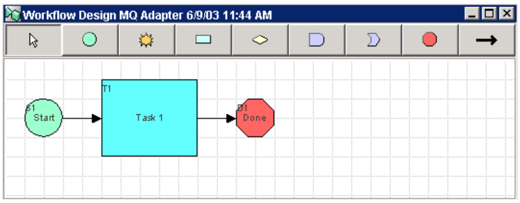
7. Do one of the following:

   - To specify the expiry date for the workflow, select the Expiry check box, and from the drop-down calendar, select the desired date. Click OK.
   - To retain the default expiry date, click OK.

   The Template Definition dialog box closes, and the Template Definition is created inside the newly created template folder, displaying the creation date and time.

   The Workflow Design window appears in the right pane.

Figure 2-18 Workflow Design Window



Workflow Design MQ Adapter 6/9/03 11:44 AM

8. Test the service or event.

   - To learn about testing services, see “Test Services in WebLogic Integration Studio” on page 2-24.
   - To learn about testing events, see “Test Events in WebLogic Integration Studio” on page 2-28.

Test Services in WebLogic Integration Studio

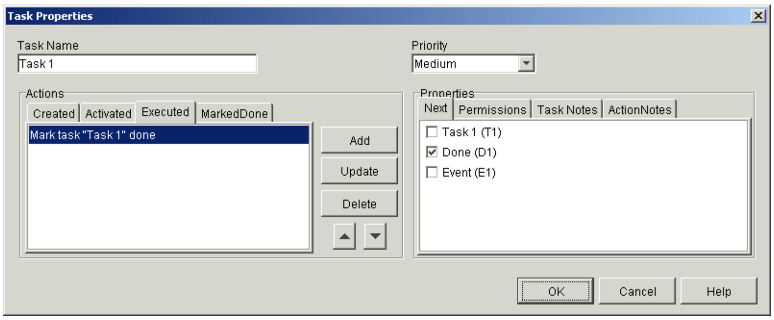
Before you test services, you must have done the following:

- Created the necessary Request Documents for the services that you want to test.
- Created and defined templates in the WebLogic Integration Studio.
- Created relevant variables which are required to execute the workflow.

To test an application view's services, do the following:

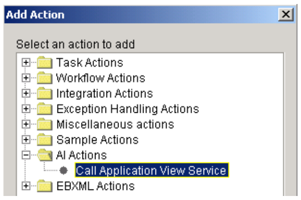
1. On the Workflow Design window, right-click Task 1 and select Properties. The Task Properties dialog box appears.
2. Select the Executed tab.

Figure 2-19 Task Properties Dialog Box



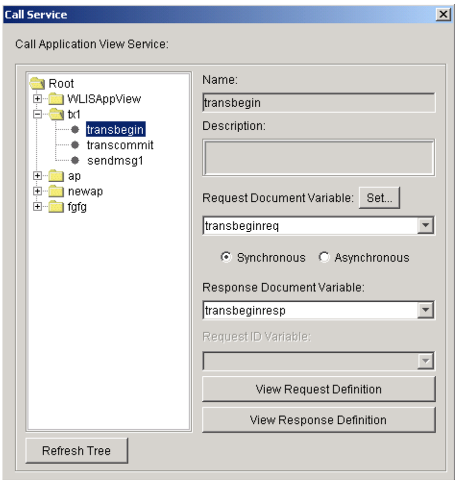
3. Click Add. The Add Action dialog box appears.

Figure 2-20 Add Action Dialog Box



4. Double-click the AI Actions folder, select Call Application View Service, and click OK. The Call Service dialog box appears.

Figure 2-21 Call Service Dialog Box



Name: transbegin

Description:

Request Document Variable: transbeginreq

(● Synchronous ○ Asynchronous)

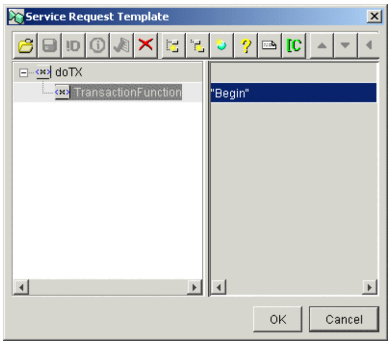
Response Document Variable: transbeginresp

Request ID Variable:

5. Select the service that you want to add. The name of the selected service appears in the Name field.

6. Click Set to set the Request Document Variable. The Service Request Template dialog box appears.

Figure 2-22 Service Request Template Dialog Box

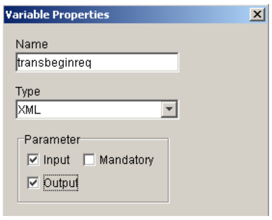


7. Click the Open Folder icon on the toolbar and select the Request Document you created. Click Yes to save the XML file to an XML template. The Service Request Document is added in the form of a template.

8. Click OK. The Service Request Template dialog box closes.

9. Select the relevant variable from the Request Document Variable list. If there are no variables, create a new variable using the Variable Properties dialog box.

Figure 2-23 Variable Properties Dialog Box



Variable Properties

Name: transbeginreq

Type: XML

Parameter
- [✓] Input
- [ ] Mandatory
- [✓] Output

10. In the Call Service dialog box, do one of the following:

    - To process the next service only after receiving the response of the first service processing, click Synchronous and select the Response Document Variable from the drop-down list.
    - To process the next service without waiting for the response of the first service processing, click Asynchronous and select the Request ID Variable from the drop-down list.

    For details on Synchronous and Asynchronous processing, see "Services" on page 1-3.

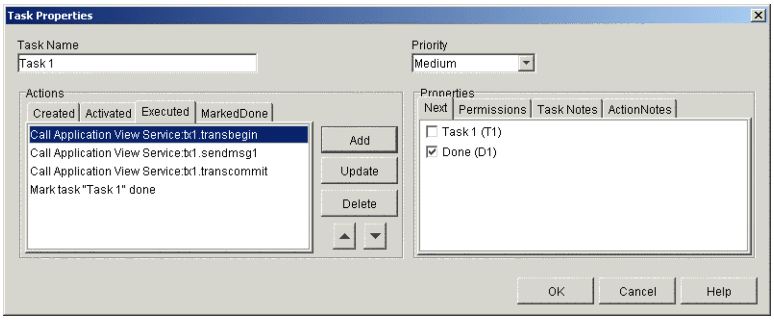
    Note: If there are no variables, create a new variable and select it.

11. Click OK. The Call Service dialog box closes, and the Task Properties dialog box appears.

    Note: Repeat steps 3 to 11 to add other services to the Task node.

12. Select the required tasks and use the up and down arrows to arrange them in sequence for execution. You can change the order depending upon the workflow requirements. The arranged tasks are shown in the following figure.

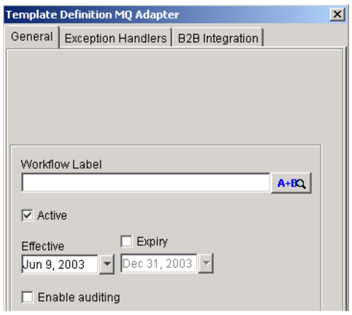
Figure 2-24 Task Properties Dialog Box with Actions



13. Click OK. The Task Properties dialog box closes.

14. In the left pane of WebLogic Integration Studio, right-click the Template Definition and select Properties. The Template Definition dialog box appears.

Figure 2-25 Template Definition Dialog Box with Active Check Box



15. Select the Active check box, and click OK. The Template Definition is activated and the Template Definition dialog box closes.

Caution: To make changes to the template, do the following:

a. Open the Template Definition dialog box, deselect the Active check box and click OK. The Template Definition is deactivated and the Template Definition dialog box closes.

b. Make changes to the Template, open the Template Definition dialog box again, select the Active check box, and click OK. The Template Definition is activated and the Template Definition dialog box closes.

This is to ensure that there is only one workflow process per active template at any given time.

16. In the left pane of WebLogic Integration Studio, right-click the Template Definition and select Save.

    Note: An asterisk before the definition name indicates that the changes to that folder have not been saved.

17. To test the services in a workflow, open the WebLogic Integration Worklist, select the tasks and execute the workflow. The tasks when executed successfully, indicate that they are functioning appropriately.

    For more information about the WebLogic Integration Worklist, see *Using the WebLogic Integration JSP Worklist* at

    http://e-docs.bea.com/wli/docs70/jspwlist/index.htm

Once the services are tested successfully, you can start using them in workflows.

Test Events in WebLogic Integration Studio

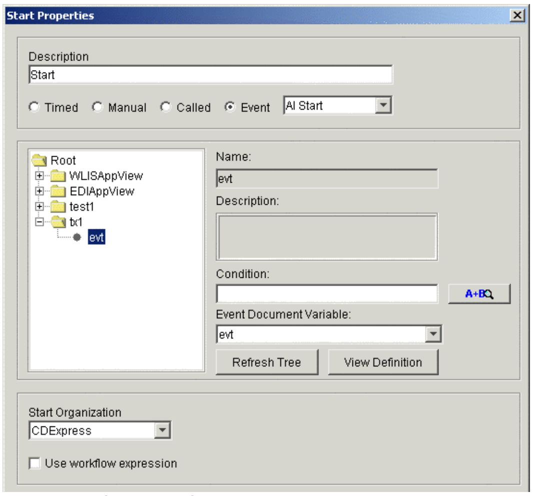
Before you test event, you must have done the following:

- Created and defined templates in the WebLogic Integration Studio.
- Created relevant variables which are required to execute the workflow.

To test an application view's events, do the following:

1. In the Workflow Design window, right-click the Start node, and select Properties. The Start Properties dialog box appears.
2. Click Event, and select AI Start from the Event drop-down list.

Figure 2-26 Start Properties Dialog Box



The left side of the dialog box displays all the application views along with their corresponding events that have been defined in the WebLogic Application Integration.

3. Select the event you want to test. The name appears in the Name field.

4. Select an Event Document Variable from the drop-down list.

   Note: If there are no variables, create a new variable and select it.

5. Select the relevant organization from the Start Organization drop-down list, and click OK. The Start Properties dialog box closes.

   On the Workflow Design window, the Start Node indicates the action setting, as shown in the following figure.

Figure 2-27 Start Node with Event Action

6. In the left pane of WebLogic Integration Studio, right-click the Template Definition and select Properties. The Template Definition dialog box appears.

7. Select the Active check box, and click OK. The Template Definition is activated and the Template Definition dialog box closes.

Caution: To make changes to the template, do the following:

a. Open the Template Definition dialog box, deselect the Active check box and click OK. The Template Definition is deactivated and the Template Definition dialog box closes.

b. Make changes to the Template, open the Template Definition dialog box again, select the Active check box, and click OK. The Template Definition is activated and the Template Definition dialog box closes.

This is to ensure that there is only one workflow process per active template at any given time.

8. In the left pane of WebLogic Integration Studio, right-click the Template Definition and select Save. The event is activated.

   Note: An asterisk before the definition name indicates that the changes to that folder have not been saved.

An event is confirmed to be functioning properly if the required messages entering the Queue are picked up by the Event Manager in the Workflow Instances.

Once the events are tested successfully, you can start using them in workflows.

For more information about using WebLogic Integration Studio, see *Learning to Use BEA WebLogic Integration* at http://edocs.bea.com/wli/docs70/bpmtutor/index.htm

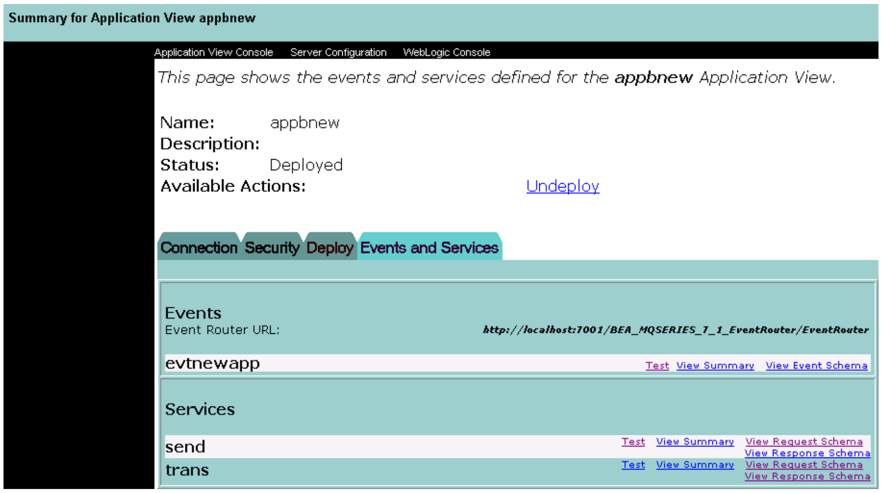
Editing an Application View

After you deploy an application view, you may want to edit it. To do this, you must undeploy the application view, reconfigure the connection parameters, and add, remove, or edit services and events.

To edit an existing application view:

1. Open the WebLogic Integration - Application View Console, and click the deployed application view. The Summary for Application View page appears.

Figure 2-28 Summary for Application View



Summary for Application View appbnew

This page shows the events and services defined for the ***appbnew*** *Application View.*

Name:	appbnew
Description:	
Status:	Deployed
Available Actions:	Undeploy

Events
Event Router URL: ***http://localhost:7001/BEA_MQSERIES_7_1_EventRouter/EventRouter***

evtnewapp	Test View Summary View Event Schema

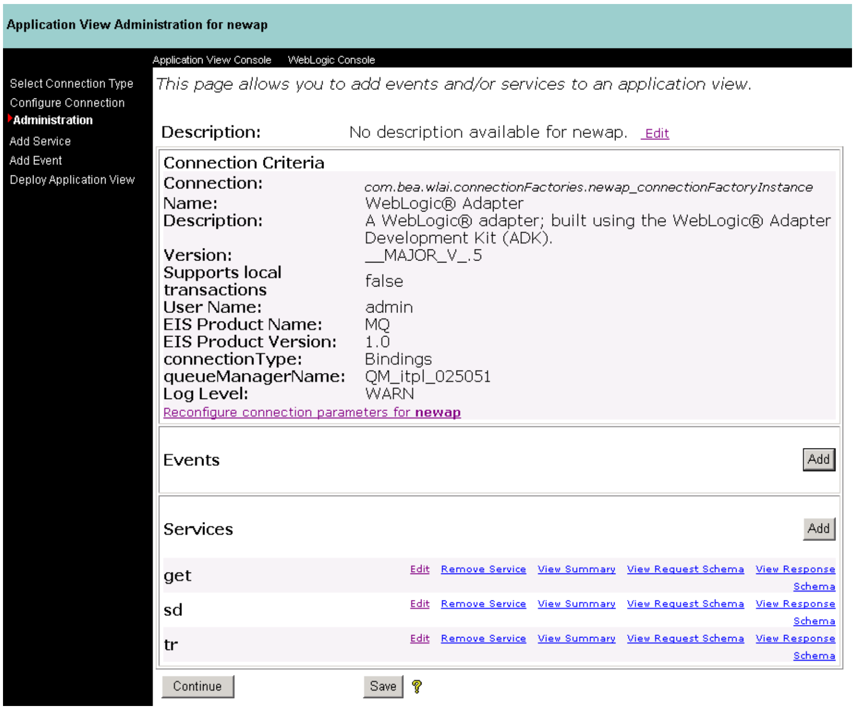
Services

send	Test View Summary View Request Schema View Response Schema
trans	Test View Summary View Request Schema View Response Schema

2. Click Undeploy and click Confirm. The Summary for Application View page (with Edit/Remove options) appears.

3. Click Edit. The Application View Administration page appears.

Figure 2-29 Application View Administration



This page allows you to add events and/or services to an application view.

Description: No description available for newap. Edit

Connection Criteria

Connection:	*com.bea.wlai.connectionFactories.newap_connectionFactoryInstance*
Name:	WebLogic® Adapter
Description:	A WebLogic® adapter; built using the WebLogic® Adapter Development Kit (ADK).
Version:	__MAJOR_V_.5
Supports local transactions	false
User Name:	admin
EIS Product Name:	MQ
EIS Product Version:	1.0
connectionType:	Bindings
queueManagerName:	QM_itpl_025051
Log Level:	WARN

Reconfigure connection parameters for **newap**

Events

Services

get	Edit	Remove Service	View Summary	View Request Schema	View Response Schema
sd	Edit	Remove Service	View Summary	View Request Schema	View Response Schema
tr	Edit	Remove Service	View Summary	View Request Schema	View Response Schema

4. Do one of the following:
   - To reconfigure the application view's connection parameters, click the Reconfigure connection parameters link.
   - To add services or events, click Add in the Service/Event row.
   - To remove a service or an event, click the Remove Service/Remove Event link.
   - To edit a service or an event, click the corresponding Edit link, modify the details on the relevant page and click Apply.
5. On the Application View Administration page, click Continue. The Deploy Application View page appears.
6. Click Deploy. The application view is deployed and the corresponding details are shown in the Summary for Application View page.

CHAPTER

3 Using the Adapter for MQSeries

This section provides information on using the Adapter for MQSeries. It contains detailed information on the services and events of the adapter.

It is organized as follows:

- About TCP/IP Connections
- About Transaction Services
- About Content Filtering in Events
- Sending and Receiving Messages

About TCP/IP Connections

This section described options available when you use a TCP/IP connection. It includes the following topics:

- Implementing User Exits
- Using CCSID in MQ Connection

Implementing User Exits

The Adapter for MQSeries provides an option to implement user exits. This is applicable only to TCP/IP MQ Connections. The available exits are:

- Send Exit: processes the message before sending it to a queue.
- Receive Exit: processes the message after getting it from a queue.
- Security Exit: performs a security check on the message.

The corresponding Java MQSeries exit interfaces that should be implemented are: MQSendExit, MQReceiveExit, and MQSecurityExit.

These exits can be configured to the MQ Connection when a TCP/IP MQ Connection is established for the adapter. During design-time, you can choose one or more of the available exits. For each exit selected, you should provide an ExitAppName. This ExitAppName should be a fully qualified Name of the Class that implements the corresponding Java MQSeries Exit interface. It should also be ensured that this Class is available in the CLASSPATH of the Application Server on which the Adapter for MQSeries is deployed.

For more information on these interfaces, see your MQSeries documentation.

To implement a user exit:

Note: This is an example on implementing Send 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` for the class file, for example `senduserexit.jar`.
3. Set the CLASSPATH for this `JAR`.

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

   http://edocs.bea.com/wladapters/mq/docs71/index.html
4. In the TCP/IP MQ Connection Page when you choose Send Exit, provide the fully qualified class name of the User Exit.
   For example, `com.bea.UserExit.UserSendExit`.

Using CCSID in MQ Connection

CCSID defines the character set of character data in the message.While obtaining an MQ Connection for the adapter's services and events, you can set a `CCSID` to establish the MQ Connection. This option is available only for TCP/IP MQ Connection.

When you get a message from a queue, you should compare the value of the `CodedCharSetId` field with the value that your application is expecting. If the two values differ, you may need to convert any character data in the message or use a data-conversion message exit if one is available.

To view the available CCSID values, click the CCSID Catalog link on the TCP/IP Type Connection page and event configuration page.

Note: While selecting a `CCSID`, ensure that the selected CCSID has a language support in the operating system on which the MQ Server is installed.

For more information on CCSID, see your MQSeries documentation or contact the MQ Administrator.

About Transaction Services

The transaction service in the Adapter for MQSeries, exposes transaction handling capabilities to the invoking Client Application. This capability makes it easier to maintain a workflow within which a sequence of service invocations on the Adapter can be done to complete a business process. It is mandatory that all the Send and Get services are part of an existing Transaction Scope. You can create such a transaction scope by using the transaction service.

The transaction service provides three options:

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

Note: The transaction boundary services (Begin and Commit/BackOut) are mandatory for sending and receiving messages. A SendMessage service by itself, will never result in a message being placed on the queue.

During run-time, one of these values can be given in the TransactionFunction tag in the Request Document for a transaction service.

Note: The invoking client application manages the transaction. If an exception occurs at runtime, the client application should call a Transaction BackOut to roll back the changes that might have occurred since the start of the current Transaction scope.

You invoke a service with a transaction scope. Transaction Begin is the first service, and Transaction Commit is the last, and you can have as many services as you want in the middle.

While executing Begin Transaction, the application checks if a Transaction Scope already exists. If it does, it throws a `javax.resource.ResourceException`. When any other service is invoked, the application checks if a Transaction Scope exists. If it does not, it throws a `javax.resource.ResourceException`, and the service invocation is terminated.

If the service execution is not successful again, a `javax.resource.ResourceException` is thrown, and the service invocation is terminated.

Each of the SendMessage, SendRequest, and GetMessage service types can be invoked within a Transaction Scope iteratively, in any sequence.

If the Sequence of Executions are successful, the Client can invoke a transaction commit service to save the process changes permanently.

If a ResourceException is thrown during the Execution process, the Client can invoke a transaction backout to roll back the changes that happened since the start of the current Transaction Scope.

About Content Filtering in Events

One of the features of the Adapter for MQSeries is the Content Filtering option, available in events. Using this feature, you can filter the messages in a Queue, based on the contents of the message.

You can do this by specifying an optional payload filter, which consists of a string of characters, including case-sensitive wildcards. Upon receiving a message, you can consume the message off the queue (delete it) or merely read it and leave it on the queue (known as browsing). You can set the queue and the queue manager destination parameters from whom you receive messages.

While configuring an event, select Yes for the Content Filtering option. If you select this, you should provide a fully qualified name of a Class that extends the `com.bea.adapter.mqseries.AbstractContentFilter` Class. This Class is available in `contentfilter.jar` which can be extracted from the `EAR` file of the Adapter for MQSeries.

You should overwrite the `Abstract matchContent` function in this Abstract Class by implementing the Content Filtering logic. You should also ensure that this Class is available in the `CLASSPATH` of the Application Server on which the Adapter for MQSeries is deployed.

At the time of executing the event, the message obtained from the Queue is passed on to this function as an input and depending upon the Boolean value returned by this function, the application creates an event for this message.

Such an event is an `XML` document describing the `MQMD` attributes of the message and the application data.

To implement Content Filtering for an event, do the following:

1. Extract the `contentfilter.jar` from the `EAR` of the Adapter for MQSeries and include it in the environment where the Content Filtering Code will be developed.
2. Create a Content Filter class extending it to `com.bea.adapter.mqseries.AbstractContentFilter.`
3. Create a Sample Content Filter Class Code:

```
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;
}
```

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 "Set the CLASSPATH" in *BEA WebLogic Adapter for MQSeries Installation and Configuration Guide* (for WebLogic Integration 7.0 or 2.1, as applicable).
6. Open Command Prompt to start the Server.
7. While configuring the event in the application view, choose Content Filtering.

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.

Sending and Receiving Messages

This section discusses in detail, the features of the adapter that you can use while sending and receiving messages. These features are classified under the following headings:

- Sending a Datagram Message
- Sending a Reply Message
- Sending Messages to Remote Queues
- Sending Group Messages
- Sending Messages with MQRFH2 Header Information
- About Receiving Group Messages
- Using Data Formats in Services and Events
- Handling MQ Message Descriptor Values in Services and Events
- Handling Request, Response Documents and MQMD Ids
- Handling Errors and Exceptions

Sending a Datagram Message

You can send a Datagram message using the SendMessage service.

Segmentation is allowed only while sending a Datagram message, to allow the Queue Manager manage the segmentation of large messages. This is same for sending a Reply and a Request message.

Before sending a Datagram message, do the following in the Request Document:

- Configure the Message Descriptor data that you want to override. For details, see “Handling MQ Message Descriptor Values in Services and Events” on page 3-16.
- Under the Data tag, do the following:
  - Provide the required format: TEXT/Binary/XML. You can provide XML only if you selected it during design-time.
  - Provide the Application data that you want to send to the Queue. This is mandatory. The application data should not be empty.

You can use the same SendMessage service to send as many number of Datagram messages as the Destination Queue can hold.

For details on the Destination Queue’s message holding capacity, contact your MQ Server Administrator.

Sending a Reply Message

You can send a Reply message using the SendMessage service.

Segmentation is allowed only while sending a Reply message, to allow the Queue Manager manage the segmentation of large messages. This is same for sending a `Datagram` and a Request message.

Before sending a Reply message, do the following in the Request Document:

- Configure the Message Descriptor data that you want to override. For details, see “Handling MQ Message Descriptor Values in Services and Events” on page 3-16.
- Under the Data tag, provide the required format: `TEXT/Binary/XML`. You can provide `XML` only if it was selected during design-time.
- Provide the `CorrelationId` of the Reply message. This is mandatory and must be given as part of the Request Document.
- Provide the `MessageId`. It is optional. If you do not provide the `MessageId`, the `MessageId` of the reply message is generated by the queue manager. This Id will be a hexadecimal value in String representation.

- Provide the `MQRFH2` data if applicable. For more information, see "Sending Messages with MQRFH2 Header Information" on page 3-13.
- Provide the application data that you want to send to the queue. This is mandatory. The application data should not be empty.

You can use the same SendMessage service to send as many number of Reply Messages as possible that the Destination Queue can hold.

For details on the Destination Queue's message holding capacity, contact your MQ Server Administrator.

Sending Messages to Remote Queues

The Adapter for MQSeries provides an option of sending messages to a Remote Queue. Once an MQ Connection is obtained on a specified Queue Manager, either in Bindings mode or TCP IP mode, you can send a message to a queue that is remote to this MQ Manager. This MQ Manager must have a queue definition of the Remote Queue. Moreover, there should be a Transmission Queue, a Sender Channel configured on this Queue Manager, and a Receiver Channel configured on the Remote Queue Manager where the Remote Queue resides.

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

Sending Group Messages

A Group Message is a set of messages that follows a sequence.

The Group Messaging facility shares a common `GroupId` and a Message Sequence Number (`MsgSeqNumber`) that increments sequentially for all the messages being sent. You must provide these two inputs to send a Group Message. You have two options of doing this:

- Use the GroupId and MsgSeqNumber generated by the MQ Server for the first message in the Group's Response Document as well as for the subsequent messages in the group. Increment the MsgSeqNumber obtained from the Response of the previous group message by 1 and use it as the MsgSeqNumber for the next message in the group. Continue incrementing the MsgSeqNumber for each subsequent message.

- The second option it to provide a GroupId and MsgSeqNumber of your choice for all the messages in the group. GroupId should remain the same for all subsequent messages but increment the MsgSeqNumber by 1 for each subsequent message.

  Note: While using this option, ensure that the GroupId used is unique for this group within the queue, to which the messages are sent.

The Adapter for MQSeries provides an option to send Group Messages using SendMessage and SendRequest services.

The Request Schema of these services has the following tags which 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>
```

The GroupMessageOptions tag is Optional in the Request Document and becomes mandatory when a Group Message is opted.

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

- Sending the First Message: While sending the First message in the Group to the Queue, the value of `IsFirstMessage` should be `True`.

  The `GroupId` is optional, but if the value is provided, it is used as the `GroupId` for all subsequent group Messages in the Queue. If not, the MQ Server generates a value and it is returned to the user in the Response Document. Similarly, `MsgSeqNumber` is another element that is optional for the First Message. This value or the MQ Server generated value is returned to the user in the Response Document.

- Sending Intermediate messages: While sending an Intermediate message in the Group to the Queue, the value of `IsIntermediateMessage` should be `True`.

  The `GroupId` is a mandatory element and should be the value that was returned in the Response Document after the First message was sent. Otherwise, a New Group will be created for this Message. Similarly, `MsgSeqNumber` is also a mandatory element and should be the incremented value of the `MsgSeqNumber` that was returned in the Response Document of the previous message in the group.

- Sending the Last message: While sending the Last message in the Group to the Queue, the value of `IsLastMessage` needs to be `True`.

  The `GroupId` is a mandatory element and should be the value that was returned in the Response Document after the First message was sent. Otherwise, a New Group will be created for this Message. Similarly, `MsgSeqNumber` is also a mandatory element 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 in a Workflow

You can send group messages in two different ways, by:

- Executing Services Programmatically in a Workflow

  You can 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. In such case, the workflow requires an additional logic to pick the GroupId and MsgSeqNumber from the first message’s Response Document and set the GroupId as input in the subsequent SendMessage and SendRequest services’ Request Documents. However, the MsgSeqNumber should be an incremented value of the previous message’s MsgSeqNumber.

- Executing Services from a Worklist

  You specify the GroupId and MsgSeqNumber within the GroupId and MsgSeqNumber tags respectively. You do this within the Request Document of the first message and also all the subsequent messages. You should ensure that the GroupId used is unique for the group within the queue to which the messages are lined up.

Executing Services Programmatically in a Workflow

To execute services programmatically in a workflow, do the following:

1. Determine the type of message that you want to send (`Datagram/Reply/Request`).
2. Create an application view. For details, see Chapter 2, "Defining Application Views for the Adapter for MQSeries."
3. Add a Transaction service to start and stop the Transaction Scope during execution.
4. Add the required services, for example, SendMessage.
5. Deploy the application view.
6. Create a Template in the WebLogic Integration Studio.
7. Add the configured services to the Task node in the sequence required. The sequence should always start with a Transaction Begin and end with a Transaction Commit or Transaction BackOut.
8. Provide an additional logic which will pick the GroupId and MsgSeqNumber from the first message's Response Document and set the same GroupId in the subsequent service's Request Documents, as input. The MsgSeqNumber should be an incremented value of the previous message's MsgSeqNumber.

   Note: In this case, do not provide the GroupId and MsgSeqNumber for the first message in the group.
9. Create the Request Documents corresponding to each message in the group.
10. Invoke the added service iteratively, and for each iterative call, associate the message in the Request Document with the corresponding GroupId, MessageSequence Number, and MessageId specified in their respective tags.

The group messages are sent when you execute the workflow.

Executing Services from a Worklist

To execute services directly from a worklist, do the following:

1. Determine the type of message that you want to send (`Datagram/Reply/Request`).
2. Create an application view. For details, see Chapter 2, “Defining Application Views for the Adapter for MQSeries.”
3. Add a transaction service to start and stop the Transaction Scope during execution.
4. Add the required services, for example, SendMessage.
5. Deploy the application view.
6. Create a Template in the WebLogic Integration Studio.
7. Add the configured services to the Task node in the sequence required. The sequence should always start with a Transaction Begin and end with a Transaction Commit or Transaction BackOut.
8. Provide the GroupId and MsgSeqNumber of your choice. You should ensure that the GroupId is unique for this group and is used for the first message and also all the subsequent messages. However, for the MsgSeqNumber of each subsequent message, you have to provide a new MsgSeqNumber, with an incremented value of the previous message’s MsgSeqNumber.
9. Create the Request Documents corresponding to each message in the group.
10. Invoke the added service iteratively, and for each iterative call, associate the message in the Request Document with the corresponding GroupId, MessageSequence Number, and MessageId specified in their respective tags.

The group messages are sent when you execute the workflow.

Sending Messages with MQRFH2 Header Information

The Adapter for MQSeries provides an option to include `MQRFH2` Header Information along with application data. You can do this in SendMessage and SendRequest services.

To use this option, you must configure the service at design-time with a `MQRFH2` format and the `MQRFH2` Header Information should be provided in the following structure:

```
<MQRFH2>
```

```
   <Encoding></Encoding>
   <CodedCharSetID></CodedCharSetID>
   <Format></Format>
   <NameValueCCSID></NameValueCCSID>
   <NameValueDatan></NameValueDatan>

</MQRFH2>
```

The `MQMD` format can also be overridden during run-time to `MQRFH2`. If this is done, then the `MQRFH2` Header Information has to be provided in the Request Document with valid data for the Format tag under Message Descriptor tag.

While retrieving messages bearing `MQRFH2` Header Information, the application by default receives the `MQRFH2` Header Information and the Application Data separately, and provides them as part of the Response Document. For details, see the Response Schema under “GetMessage”.

About Receiving Group Messages

The GetMessage service and the events can be used to receive group messages from Queue.

In Get Message, the `GroupId` of the required message should be provided in the Request Document. The First Message in that Group is received and returned to the User.

In events, all the messages in the queue are received and given back to the invoking application.

When Content Filtering is opted, only those messages are received that meet the filtering criteria.

Using Data Formats in Services and Events

The Adapter for MQSeries supports TEXT, Binary, and XML stat formats.

The following conditions apply in using these data formats.

- TEXT – This format can be changed to binary at run-time.
- Binary – This format can be changed to text at run-time.

- XML – During design-time, you should provide the schema for the XML application data. This data format can be opted during run-time only if it was opted during design-time.

SendMessage and SendRequest XML Data Formats

This XML application data needs to be given within the contents tag of the Request Document. For example, consider the following XML application data:

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

<root>
<elm1>
<elm2>abcd</elm2>
</elm1>
<elm3>efg</elm3>
</root>
```

The Request Document format, which should contain the XML application data, will have the following format:

```
<?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>
              <root>
<elm1>
<elm2>abcd</elm2>
</elm1>
<elm3>efg</elm3>
</root>
</Content>
</Data>
</ServiceName>
```

If the `XML` Application data is not provided within the Request Document format as mentioned, the application will throw an exception and the service will not be executed.

The ServiceName tags must be the name of the service you are working with. The XML inside the content tag must match the XML schema you enter while designing a service with XML Data Format.

GetMessage XML Data Formats

You can configure a GetMessage service at design-time, in one of the mentioned Data formats. Additionally, you can also change this Data format to any of the remaining two Data formats in the Request Document, during design-time.

Event XML Data Formats

You can configure an event at design-time, in one of the mentioned data formats.

Handling MQ Message Descriptor Values in Services and Events

You can use MQ Message Descriptor (`MQMD`) attributes when you configure services and events. In the SendMessage and SendRequest services, at design-time, you can configure the message descriptor attributes for the message. The Request Schema of SendRequest and SendMessage has a MessageDescriptor tag.

The `MQMD` attribute values that can be overridden at run-time are as follows:

- ExpirationPolicy
- Priority
- PersistPolicy
- CharacterSet
- Format

The Response Document for GetMessage service and the event document provides a detailed description of the `MQMD` attribute values.

For more information, see the corresponding Request Schema and Response Schema for each service.

Note: If any of the `MQMD` values are either wrongly mentioned or omitted (both design-time and run-time), the application will take the default values.

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

Handling Request, Response Documents and MQMD Ids

You should create a Request Document with the required run-time attributes and passed as an input during each service invocation. You should ensure that the Request Document is in full accordance with the Request Schema for the specific service. Otherwise, the application will throw an exception and the invocation will be aborted.

There is no Request Document for events. For each event created, an event document is generated by the application. This has a `MQMD` part where the message descriptors are provided and a data part where the application data and header information are provided.

The integration examples given in this document provide the Request Document that use the service name as the Root element. This is not mandatory. The application accepts any well formed document with any Root element name and valid values to execute a service.

A Response Document is generated by the application for each service.

Note: You can view the Request Schema and the Response Schema for each service on the Summary for Application View page.

MQMD Ids such as `MessageId`, `CorrelationId`, `GroupId` are hexadecimal values in String representation. When providing an Id in the request document, ensure that it is valid.

Handling Errors and Exceptions

If an MQ Exception occurs during the execution of a service, the Application builds a Response Document with Status - FAILURE. Additionally, an Error tag displays the MQ Reason Code for the Exception, Completion Code and MQ Description for the MQ Reason Code.

For Exceptions other than MQ Exceptions, a `javax.resource.ResourceException` is thrown back to the invoking Exception. 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.

CHAPTER

4 Example of Using Services and Events

This section illustrates how the services and events of the Adapter for MQSeries function in a given scenario. The illustrations focus on the parameters that are specified for each service and event during design-time, the corresponding request documents used, and the related response documents that are generated during run-time, after the workflow is executed in the WebLogic Integration Studio.

It contains the following topics:

- Service Example
- Event Example

The information provided here assumes that you have in-depth knowledge of Workflow Design, Workflow Templates, Worklists, and WebLogic Integration Studio.

If you do not have the required knowledge of workflows or the WebLogic Integration Studio, see the following documents:

- *Using the WebLogic Integration Studio* at http://edocs.bea.com/wli/docs70/studio/index.htm
- *Learning to Use BPM with WebLogic Integration* at http://edocs.bea.com/wli/docs70/bpmtutor/index.htm

Service Example

The service example involves a scenario where a set of services are invoked as part of a transaction scope from within a workflow instance, in a given sequence:

Transaction Begin→SendMessage→SendRequest→GetMessage→Transaction Commit.

The information in the following paragraphs, contains the configuration details of each service during design-time, in the given sequence, and the corresponding Request Document and Response Document formats generated during run-time. The Request Documents and Response Documents contain the configuration details that you set during run-time.

Note: The Response Document is generated only after a workflow is executed, using the WebLogic Integration Studio.

Transaction Service - Begin

Set the following values for the Transaction service in the Transaction option. For details, see "About Transaction Services" on page 3-4.

Field Name	**Value**
Unique Service Name	`TransactionBegin`
Description	`To start a Transaction Scope.`

The Request Document for this service and the corresponding Response Document generated by the adapter are as follows:

Request Document

```
<?xml version="1.0" encoding="UTF-8"?>
<TransactionBegin>
<TransactionFunction>Begin</TransactionFunction>
</TransactionBegin>
```

Response Document

```
<?xml version="1.0" encoding="UTF-8"?>
<TransactionBegin>
<Result>
<Status>SUCCESS</Status></Result></TransactionBegin>
```

SendMessage

Set the following values for the Send Message service in the SendMessage option.

Field Name	Value
Unique Service Name	`SendPlainMessage`
Service Type	`SendMessage`
Description	`Send a Datagram Message`
Message Type	`Datagram`
Queue Name	`Queue_0251`
Expiration Policy	`6000`
Message Priority	`7`
Persistence Policy	`NotPersistent`
Character Set	`819`
Message User	`admin`
Segmentation Policy	`NotAllowed`

Field Name	Value
Report Messaging Options - COA	None
Report Messaging Options - COD	None
Report Messaging Options - Exception	None
Report Messaging Options - Expiration	None
Format	None
Data Format	TEXT

The Request Document for this service and the corresponding Response Document generated by the adapter are as follows:

Request Document

```
<?xml version="1.0" encoding="UTF-8"?>
<SendPlainMessage>
<MessageDescriptor></MessageDescriptor>
<Data>
<Format>TEXT</Format>
<Content>hello world</Content>
</Data>
</SendPlainMessage>
```

Response Document

```
<?xml version="1.0" encoding="UTF-8"?>
<SendPlainMessage>
<Result>
<MessageId>414D5120514D5F746174615F746F77659472C83E12600000</Mess
ageId>
<Status>SUCCESS</Status>
</Result></SendPlainMessage>
```

SendRequest

Set the following values for the SendRequest service in the SendRequest option.

Field Name	Example
Unique Service Name	`SendRequestMessage`
Service Type	`SendRequest`
Description	`Send a Datagram Request`
Message Type	`SendRequest`
Local/Remote Queue Name	`Queue_02545`
Expiration Policy	`6000`
Message Priority	`5`
Persistence Policy	`NotPersistent`
Character Set	`819`
Message User	`admin`
Segmentation Policy	`Notallowed`
Report Messaging Options - COA	`None`
Report Messaging Options - COD	`None`
Report Messaging Options - Exception	`None`
Report Messaging Options - Expiration	`None`
Reply To Queue Name	`Queue_reply`
Format	`None`
Data Format	`TEXT`

The Request Document for this service and the corresponding Response Document generated by the adapter are as follows:

Request Document

```
<?xml version="1.0" encoding="UTF-8"?>
<SendRequestMessage>
<MessageDescriptor></MessageDescriptor>
<Data>
<Format>TEXT</Format>
<Content>hello world</Content>
</Data>
</SendRequestMessage>
```

Response Document

```
<?xml version="1.0" encoding="UTF-8"?>
<SendRequestMessage>
<Result>
<MessageId>414D5120514D5F746174615F746F77659472C83E22600000</Mess
ageId>
<Status>SUCCESS</Status>
</Result></SendRequestMessage>
```

GetMessage

Set the following values for the GetMessage service in the GetMessage option.

Field Name	Example
Unique Service Name	`GetMessage`
Service Type	`GetMessage`
Description	`Getting the Message`
Queue Name	`Queue_02554`
Message Consumption	`Browse`
Time Out	`6000`
Data Format	`TEXT`

The Request Document for this service and the corresponding Response Document generated by the adapter are as follows:

Request Document

```
<?xml version="1.0" encoding="UTF-8"?>
<GetPlainMessage>
<MessageId>414D5120514D5F746174615F746F77659472C83ED2400000</Mess
ageId>
<DataFormat>TEXT</DataFormat>
</GetPlainMessage>
```

Response Document

```
<?xml version="1.0" encoding="UTF-8"?>
<GetPlainMessage>
<Result>
<Status>SUCCESS</Status>
<GetInfo>
<QueueName>test</QueueName>
<MessageConsumption>Browse</MessageConsumption>
</GetInfo>
<PayLoad>
<MQMD>
<MessageType>Datagram</MessageType>
<MessageId>414D5120514D5F746174615F746F77659472C83ED2400000</Mess
ageId>
<CorrelationId></CorrelationId>
<GroupId></GroupId>
<Format>MQSTR</Format>
<ReplyToQueueName></ReplyToQueueName>
<ReplyToQueueManagerName>QM_tata_tower1</ReplyToQueueManagerName>
<UserId>Administrato</UserId>
<ApplicationIdData></ApplicationIdData>
<PutApplicationName>C:\WINNT\System32\MMC.EXE</PutApplicationName
>
<PutDateTime>21/5/2003 - 11:48:59</PutDateTime>
<ApplicationOriginData></ApplicationOriginData>
</MQMD>
<Message>
<MQRFH2_Contents></MQRFH2_Contents>
<Data>
<Content>Hello world</Content></Data>
```

```
</Message>
</PayLoad>
</Result></GetPlainMessage>
```

Transaction Service - Commit

Set the following values for the Transaction service in the Transaction option.

Field Name	Value
Unique Service Name	`TransactionCommit`
Description	`To end a Transaction Scope`

The Request Document for this service and the corresponding Response Document generated by the adapter are as follows:

Request Document

```
<?xml version="1.0" encoding="UTF-8"?>
< TransactionCommit >
<TransactionFunction>Commit</TransactionFunction>
</TransactionCommit>
```

Response Document

```
<?xml version="1.0" encoding="UTF-8"?>
<TransactionCommit>
<Result>
<Status>SUCCESS</Status></Result></TransactionCommit>
```

Event Example

The event example involves a scenario where an event is configured to get activated once it receives a message in the Queue. Once an event is activated, and it successfully reads the message, an Event Response Document is generated for the message received in the Queue.

Set the following values for the event in the event option. For details, see “Step 5B. Add an Event to an Application View” on page 2-14.

Field Name	**Example**
Unique Event Name	Sample Event
Description	To receive messages
Connection Type	`TCP`
Queue Manager	`QM_sytem_0255`
Queue Manager Host	`localhost`
Queue Manager Channel	`sys_0125`
Queue Manager Port	`1414`
Queue Manager CCSID	`819`
Queue To Monitor	`Queue_MT_0051`
Message Consumption	`Browse`
Data Format	`TEXT`
Content Filter Required	`No`

The Response Document for this service generated by the adapter is as follows:

Event Response Document

```
<SampleEvent>
<EventInfo>
<QueueName>test</QueueName>
<MessageConsumption>browse</MessageConsumption>
</EventInfo>
<PayLoad>
<MQMD>
<MessageType>DATAGRAM</MessageType>
<MessageId>414D5120514D5F746174615F746F77659472C83ED2400000</Mess
ageIdD>
<CorrelationId></CorrelationId>
<GroupId></GroupId>
<Format>MQSTR</Format>
<ReplyToQueueName></ReplyToQueueName>
<ReplyToQueueManagerName>QM_tata_tower1</ReplyToQueueManagerName>
<UserId>Administrato</UserId>
<ApplicationIdData></ApplicationIdData>
<PutApplicationName>C:\WINNT\System32\MMC.EXE</PutApplicationName
>
<PutDateTime>21/5/2003 - 11:48:59</PutDateTime>
<ApplicationOriginData></ApplicationOriginData>
</MQMD>
<Message>
<MQRFH2_Contents>
<StrucId></StrucId>
<Version></Version>
<StrucLength></StrucLength>
<Encoding></Encoding>
<CodedCharSetId></CodedCharSetId>
<Format></Format>
<Flags></Flags>
<NameValueCCSID></NameValueCCSID>
<NameValueLengthn></NameValueLengthn>
<NameValueDatan></NameValueDatan>
</MQRFH2_Contents>
<Data>Hello world</Data>
</Message>
</PayLoad>
</SampleEvent>
```

A Schema Formats of Services and Events

This section contains the schema formats of the services and events of the Adapter for MQSeries. These are of two types: Request Schema and Response Schema. The Request Schema is used to prepare the Request Document and the Response Schema is used by the adapter to generate Response Document.

It contains the following topics:

- Service Schemas
- Event Schema

To view both the types of schemas, click the corresponding link on the Application View Administration page of the Application View Console.

For details on the Request Document and Response Documents for the schemas discussed here, see Chapter 4, “Example of Using Services and Events.”

Service Schemas

The Service schemas are categorized under the following headings:

- Transaction
- SendMessage
- SendRequest

- GetMessage

Transaction

The schema formats of the Transaction service are as follows:

Request Schema Format

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="send">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="TransactionFunction" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Response Schema Format

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="send">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Result">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Status" type="xsd:string"/>
<xsd:element name="Error" minOccurs="0">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="CompletionCode" type="xsd:string"/>
<xsd:element name="ReasonCode" type="xsd:string"/>
<xsd:element name="ReasonCodeDescription" type="xsd:string"/>
<xsd:element name="Trace" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
```

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

SendMessage

The schema formats of the SendMessage service are as follows:

Request Schema Format

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="sendmsg">
<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>
```

Response Schema Format

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="sendmsg">
<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" minOccurs="0">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="CompletionCode" type="xsd:string"/>
<xsd:element name="ReasonCode" type="xsd:string"/>
<xsd:element name="ReasonCodeDescription" type="xsd:string"/>
<xsd:element name="Trace" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="GroupMessage" minOccurs="0">
<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

The schema formats of the SendRequest service are as follows:

Request Schema Format

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="sendreq">
<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>
```

Response Schema Format

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="sendreq">
<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" minOccurs="0">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="CompletionCode" type="xsd:string"/>
<xsd:element name="ReasonCode" type="xsd:string"/>
<xsd:element name="ReasonCodeDescription" type="xsd:string"/>
<xsd:element name="Trace" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="GroupMessage" minOccurs="0">
<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

The schema formats of the GetMessage service are as follows:

Request Schema Format

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="getmsg">
<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>
```

Response Schema Format

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="getmsg">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Result">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Status" type="xsd:string"/>
<xsd:element name="Error" minOccurs="0">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="CompletionCode" type="xsd:string"/>
<xsd:element name="ReasonCode" type="xsd:string"/>
<xsd:element name="ReasonCodeDescription" type="xsd:string"/>
```

```
<xsd:element name="Trace" type="xsd:string"/>
</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" minOccurs="0">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="StrucId" type="xsd:string"/>
<xsd:element name="Version" type="xsd:long"/>
<xsd:element name="StrucLength" type="xsd:long"/>
<xsd:element name="Encoding" type="xsd:long"/>
<xsd:element name="CodedCharSetId" type="xsd:long"/>
<xsd:element name="Format" type="xsd:string"/>
<xsd:element name="Flags" type="xsd:long"/>
```

```
<xsd:element name="NameValueCCSID" type="xsd:long"/>
<xsd:element name="NameValueLengthn" type="xsd:long"/>
<xsd:element name="NameValueDatan" type="xsd:long"/>
</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

This is the event schema used to generate an Event Response Document. To see the Event Response Document, see Chapter 4, “Example of Using Services and Events.”

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="event">
<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"/>
<xsd:element name="Version" type="xsd:long"/>
<xsd:element name="StrucLength" type="xsd:long"/>
<xsd:element name="Encoding" type="xsd:long"/>
<xsd:element name="CodedCharSetId" type="xsd:long"/>
<xsd:element name="Format" type="xsd:string"/>
<xsd:element name="Flags" type="xsd:long"/>
<xsd:element name="NameValueCCSID" type="xsd:long"/>
<xsd:element name="NameValueLengthn" type="xsd:long"/>
<xsd:element name="NameValueDatan" type="xsd:long"/>
</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>
```

B Logging Messages

This section describes the logging for services and events. It contains the following topics:

- About Logging
- Levels of Logging
- Logging and Performance

You set the log while deploying an application view. To know where to make the settings, see “Step 6. Deploy the Application View” on page 2-16.

About Logging

Logging is an essential feature of an adapter. Most adapters are used to integrate different applications and do not interact with end-users while processing data. Unlike a front-end component, when an adapter encounters an error or warning condition, it cannot stop processing and wait for an end-user to respond.

Many business applications that are connected by adapters are mission-critical. For example, an adapter might be required to keep an audit report of every transaction with an EIS. Consequently, adapter components should provide both accurate logging and auditing information. The adapter logging framework is designed to accommodate both logging and auditing.

Levels of Logging

Logging is provided by both the BEA adapter framework and by the BEA WebLogic Adapter for MQSeries.

The BEA WebLogic Integration framework provides five distinct levels of logging:

Table B-1 Logging Level Definitions

This Level	Indicates
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.
DEBUG	Information used to determine how the adapter works internally.

Logging and Performance

The additional logging capabilities provided by the adapter are categorized and not strictly hierarchic. These loggings are designed to provide help in debugging with minimum effect on performance. All internal adapter loggings are controlled through the additional logging settings, and all additional settings route their output to the standard debug setting.

If you configure the adapter for additional settings and do not configure standard logging settings, the logs are generated but never appear in output. This affects performance, because the production of the logging continues even though you receive no benefit of the additional logging information.

APPENDIX

C Run-Time Parameter Values

The run-time parameter values of the request schema for each service are tabulated as follows:

Table C-1 Run-Time Parameter Values

Field Name	Description	Permitted Values
Transaction Service		
Transaction Function	Mandatory tag	Begin, Commit, BackOut
Send Message and Send Request Service		
Expiration Policy	Optional tag	Any integer value (1/10th of a second)
Priority	Optional tag	Any value between 0 to 9 or `AsQueueDef`
PersistPolicy	Optional tag	`Persistent`, `NotPersistent`, `AsQueueDef`
CharacterSet	Optional tag	Refer CharacterSetCatalog link on SendRequest and SendMessage page during design-time
Format	Optional tag	None, `String`, `MQRFH2`
MQRFH2	Optional tag, mandatory when the Format is `MQRFH2`	`Encoding`, `CodedCharSetId`, `Format`, `NameValueCCSID`, `NameValueDatan`

Table C-1 Run-Time Parameter Values (Continued)

Field Name	Description	Permitted Values
Encoding	Mandatory when `MQRFH2` tag is used	Integer representing a valid encoding
CodedCharSetID	Mandatory only when `MQRFH2` tag is used	Integer representing a valid `CharacterSet`
Format	Mandatory only when `MQRFH2` tag is used	Any valid format recognized by MQSeries
NameValueCCSID	Mandatory only when `MQRFH2` tag is used	Integer representing a valid `CharacterSet`
NameValueDatan	Mandatory only when `MQRFH2` tag is used	An `XML` string
GroupMessage Options	Optional tag, mandatory if Group Messaging is required	
IsFirstMessage	Optional, Mandatory tag only for first message in the group. Value should be True.	True
IsLastMessage	Optional, Mandatory tag only for Last Message. Value should be True.	True
IsIntermediate Message	Optional, Mandatory tag only for Intermediate Messages. Value should be True.	True
GroupId	Optional tag for first message. Mandatory for Intermediate and Last Message.	A valid hexadecimal `GroupId` in string format
MsgSeqNumber	Optional tag for first message. Mandatory for Intermediate and Last message.	Positive Integer
MessageID	Optional tag	A valid hexadecimal `MessageId` in string format
CorrelationID	Mandatory tag for reply message. Not required for Datagram and Request Messages.	A valid hexadecimal `CorrelationId` in string format
Data	Mandatory tag, format, and content tags are mandatory	Format and Content tags
Format	Mandatory tag	TEXT, Binary, XML
Content	Mandatory tag	Application data in `String`

Table C-1 Run-Time Parameter Values (Continued)

Field Name	Description	Permitted Values
Get Message Service		
MessageID	Optional tag	A valid hexadecimal `MessageId` in string format
CorrelationID	Optional tag	A valid hexadecimal `CorrelationId` in string format
GroupID	Optional tag	A valid hexadecimal `GroupId` in string format
Data Format	Mandatory tag	TEXT, Binary, XML

D Error Messages and Troubleshooting

This section lists error messages that you may encounter while using the BEA WebLogic Adapter for MQSeries. It describes what may 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 as follows:

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.

Error	**javax.resource.spi.EISSystemException: An exceptional condition was encountered when <username> attempted to open a connection to the EIS; Message catalog not found**
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 “Step 4. Establish an MQSeries Connection,” in Chapter 2, “Defining Application Views for the Adapter for MQSeries.”

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 “Step 4. Establish an MQSeries Connection,” in Chapter 2, “Defining Application Views for the Adapter for MQSeries.” 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 if one or more of the User Exits have been opted. If yes, check whether the App Name 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,” in Chapter 3, “Using the Adapter for MQSeries.”

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 if the event is configured to the correct Queue. 2. Check for a message in the Queue. 3. Verify if the Connection parameters are appropriate. 4. Check if the right data format is chosen. Also check if the expected Message’s data format and the one configured to, match. 5. Check if Content Filtering has been opted for. If yes, is the Content Filter Class name correct and is it available in the execution environment where the Adapter for MQSeries is deployed. 6. Check if 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 if the Service executed is a part of the Transaction Scope.
- Check if the TransactionFunction Begin was invoked earlier. If not check whether TransactionFunction Commit or BackOut was invoked before the TransactionFunction Begin.
- Ensure to match the Request Document with the corresponding Request Schema.
- Find out if the mandatory tags and their values are provided in the Request Document.
- Check if 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 received an expected message.
- Confirm the presence of 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.
- Check if the data format provided in the Service is XML. If yes, check whether the XML Application Data provided in the Request Document matches with the XML Data Schema provided while configuring the Service.
- Check if the data format in the Request Document is XML. If yes, check if it was selected while configuring the Service during design-time.
- Check if 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’ address.

For details on configuring Services, see “Step 5. Add Services and Events” on page 2-9
For details on Messaging, see “Sending and Receiving Messages” on page 3-7

Index