BEA WebLogic Adapter for SWIFT

User Guide

Release 7.0.3
Document Date: April 2003

BEA WebLogic Adapter for SWIFT **User Guide**

Part Number	**Date**
N/A	April 2003

Table of Contents

About This Document

This document explains how to configure and use the BEA WebLogic Adapter for SWIFT to integrate SWIFT messages over an MQSeries transport with BEA WebLogic Integration.

This document covers the following topics:

- Chapter 1, “Introduction,” provides an overview of the adapter’s benefits and processes.
- Chapter 2, “Creating and Configuring an Event Adapter.”describes how to create, configure, and test an event adapter.
- Chapter 3, “Creating and Configuring a Service Adapter” describes how to create, configure, and test a service adapter.
- Chapter 4, “Transforming Document Formats,” describes how you can convert documents between XML and non-XML formats using several transformation phases.
- Chapter 5, “Modifying Document Definitions,” introduces schemas and schema repositories, and provides the steps for naming a schema repository, creating a repository manifest, and creating a schema.
- Chapter 1, “Acknowledgement Handling,” describes the process of acknowledging a document after it has passed through validation.
- Chapter 1, “Using Tracing,” explains how to set tracing options for logging and debugging.
- Appendix A, “Rules System Adapter” lists error messages and suggests what action to take for each message if displayed.
- Appendix B, “Linking SWIFT Adapters to SWIFT Networks,” describes the connecting of business applications to SWIFTAlliance.

- Appendix C, "Support for AS1 and AS2 Communications" describes support for AS1 and AS2 communications.

What You Need to Know

This document is written for systems integrators who develop interfaces between SWIFT based message systems and other applications. It describes how to use the BEA WebLogic Adapter for SWIFT using the MQSeries transport and how to develop application environments with specific focus on SWIFT message integration. It is assumed that readers know Web technologies and have a general understanding of Microsoft Windows and UNIX systems.

Related Information

The BEA corporate Web site provides all documentation for WebLogic Server and WebLogic Integration. For information about these products, go to `http://e-docs.bea.com`. Documents that you may find helpful when installing the BEA WebLogic Adapter for SWIFT are:

- BEA WebLogic Adapter for SWIFT Installation and Configuration Guide
- BEA WebLogic Adapter for SWIFT Release Notes
- BEA WebLogic Adapter for File, FTP and HTTP Installation Guides
- BEA Application Explorer Installation Guide
- BEA WebLogic Server installation and user documentation, which is available at the following URL:

  `http://edocs.bea.com/more_wls.html`
- BEA WebLogic Integration installation and user documentation, which is available at the following URL:

  `http://edocs.bea.com/more_wli.html`

Contact Us!

Your feedback on the BEA WebLogic Adapter for SWIFT 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 documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Adapter for SWIFT release.

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

Convention	Item
italics	Indicates emphasis or book titles.

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

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

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

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

Convention	Item

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

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

CHAPTER

1 Introduction

This section provides an overview of the BEA WebLogic Adapter for SWIFT. It includes the following topics:

- SWIFT Adapter Components
- Overview
- Configuring and Validating SWIFT Documents

SWIFT networks carry messages between financial institutions and must originate or arrive at financial institutions in a format described by a SWIFT message standard.

The BEA WebLogic Adapter for SWIFT transforms documents into XML format and XML representations of SWIFT documents back into SWIFT format. After the information is in XML format, it can be integrated into back or front office systems using BEA WebLogic Integration and all of the BEA application and data adapters that are available from the adapter suite of products.

J2EE™ standard interfaces and protocols such as JCA™, JDBC™, and JMS™ are also supported with the BEA WebLogic Adapter for SWIFT. The same adapters can be used to obtain information that is required to populate SWIFT messages. For example, using the BEA WebLogic Adapter for RDBMS updates in an RDBMS to trigger a SQL query that returns an XML formatted answer set that can be mapped to a SWIFT message.

Data dictionaries that describe the XML format are used to enable the mapping of XML documents to SWIFT form and SWIFT messages to XML. After structural integrity has been verified during the transformation stage, the BEA WebLogic Adapter for SWIFT performs validation, using a set of rules defined in an XML formatted rules file. Where applicable, acknowledgment documents are returned to the sending application, but only if the incoming document is structurally valid. If the content validation fails, an error code is returned in the acknowledgement document when one is expected.

You can use the BEA WebLogic Adapter for SWIFT to exchange SWIFT formatted documents over any one of the supported transport protocols with WebLogic Integration to provide a tightly integrated application infrastructure. The multiprotocol support of the adapter allows for the greatest flexibility in application integration from Web-based form submission to guaranteed delivery via IBM's MQSeries. (Note that IBM has renamed MQSeries to WebSphere MQ.)

MQSeries messaging products support application integration by sending and receiving data as messages that allow business applications to exchange information across different platforms. These products account for network interfaces, assure "once only" delivery of messages, interface 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 SWIFT provides:

- *Multi-protocol support* for integration with any form of application system or standardized message handling system.
- *Message transfer* between SWIFT Message Handling Systems and WebLogic Integration.
- *Service and event adapter integration operation* providing end-to-end business process management using SWIFT formatted messages and XML schema-defined business processes.
- *Support for custom and standard SWIFT message formats* with automatic generation of transforms into a common XML business process environment.

The adapter provides pre-packaged support for standard SWIFT message formats, but does not provide out-of-the-box the ability to customize those formats. Please contact BEA professional services if you need to customize these formats.

SWIFT Adapter Components

The BEA WebLogic Adapter for SWIFT is packaged with the components required to integrate SWIFT message information into an existing systems environment. The following components are supplied with the adapter:

- *BEA WebLogic Integration Application Views*. This is the backbone of the BEA WebLogic Adapter for SWIFT. WebLogic Integration is the host for the BEA WebLogic Adapter for SWIFT and can be used to host a multitude of BEA adapters. This architecture enables easy integration spanning multiple protocols (for example, SWIFT and MQ Series) and back-end systems and/or data sources.
- *BEA Application Explorer.* BEA Application Explorer enables custom dictionary maps (or transformation templates) to be built that can be deployed to WebLogic Integration. This powerful map builder allows complex templates (that include intelligence using pre-built and custom functions) to be applied to message formats. These templates use the supplied dictionary files to describe the document and facilitate the conversion to or from XML.
- BEA WebLogic Adapter for SWIFT *kit*. The BEA WebLogic Adapter for SWIFT kit is a set of components that enables the integration of SWIFT message formats into an environment. The following components are supplied with the kit:
  - *XML schemas (.xsd)*. These describe the SWIFT messages for event and service processing.
  - *Data dictionaries (.dic)*. These describe the SWIFT format messages to the BEA WebLogic Adapter for SWIFT.
  - *Transformation templates (.xch)*. These are the maps that are created using the transformation templates configured to convert SWIFT to XML and XML to SWIFT.
  - *Sample files (.swift and .xml)*. These files are sample documents used to build the transform templates (they conform to the 2001 standards for SWIFT). These files are supplied to help you begin creating custom maps for your specific environment.
  - *Rules files (.rul)*. Rules files are XML documents that are used to apply business rules to the SWIFT XML file (either post or prior to conversion).

The rules files are built to the SWIFT green book specification and are customizable to apply customer and/or partner specific rules.

- *Code sets (.iso)*. Currency and country (ISO standard) code set tables are provided for validation using the rules engine.

Overview

The BEA WebLogic Adapter for SWIFT provides transport protocol, transformation, and document validation support. It can receive and emit SWIFT formatted documents over any supported protocol, transform the documents into XML standard format, and validate the document against a suite of SWIFT rules.

When the BEA WebLogic Adapter for SWIFT receives a document, it can be processed in a number of ways. The BEA WebLogic Adapter for SWIFT supports bidirectional transformation using XML as the intermediate format across a number of transport protocols. The adapter is supplied with templates to convert SWIFT message types to XML and XML-data type definitions (DTDs) and schemas are supplied to SWIFT format.

The SWIFT documents are described using data dictionaries supplied with the adapter. The data dictionaries conform to the SWIFT green books, and the format for describing data elements conforms to the standards in the books. These dictionaries are in XML format and can be edited to tailor messages to individual bank and/or market standards.

Receipt of SWIFT messages has been implemented as an event adapter within WebLogic Integration. Emission of SWIFT messages has been implemented as a service adapter. These adapters provide a number of processing options that you can configure with the design-time WebLogic Integration Application View Console.

- *Transformation services*. The BEA WebLogic Adapter for SWIFT automatically recognizes the SWIFT message types and converts them into the XML equivalent. The adapter provides the full suite of message formats and their associated XML Schema Definitions (`.xsd` files).
- *Document validation rules*. During the analysis of an incoming SWIFT message, the system validates the message against the associated SWIFT rule. Validation rule specifications are stored in XML files that are freely accessible in the

directory structure. Keeping each rule in an external file facilitates the maintenance of existing rules and provides an easy way to add new ones. You can also create new rules by writing custom Java code.

- *Message acknowledgment*. Message receipt and message validation may be configured to generate message validation. A validation response may be emitted on any of the supported transports.
- *Protocol emitting*. The service adapter supports emitting SWIFT messages to any of the supported transport protocols.
- *Document chaining*. You can use any service adapter to enhance the functionality of a document flow. You can chain service adapters together.

Validation

The SWIFT messages are validated in two ways. The first way is by using the dictionaries. The dictionaries are used to parse the document and validate the structure of the message type and tag structure. The second way is by the rules engine, which provides content (domain and network) validation.

The SWIFT document is validated by use of a rule file in XML format. This file is an XML document that applies pre-built rules based on the XML tag. These are customizable, and users can write their own rules to apply extra business logic.

Depending on the direction (XML to SWIFT or SWIFT to XML), the content validation occurs before or after structural validation. The following section explains how transformation and validation works for both inbound (receiving a SWIFT message) and outbound (creating a SWIFT message) processes.

Inbound

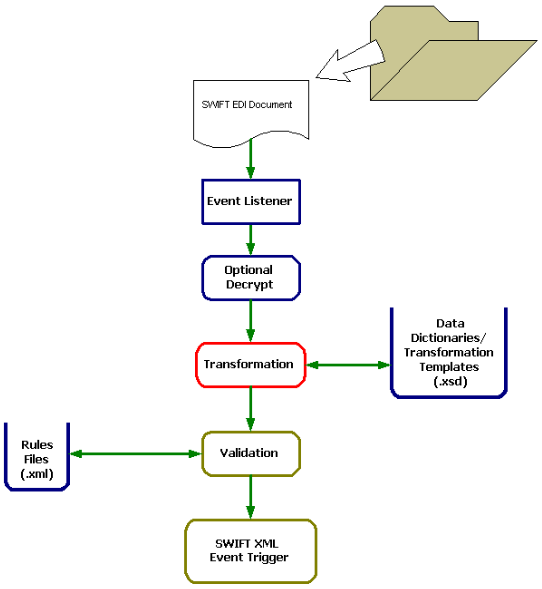
A configured listener picks up an inbound SWIFT message. The first step in the process is to apply an encryption or decryption algorithm. This is a configurable process, and you can use any Java™-based encryption processor.

The next step is the pre-parse stage. Because the iXTE maps input and output using XML format, the document (if not in XML format at this stage) must be converted to XML. This is where the pre-built (or customized) transformation templates are called to convert SWIFT format to XML.

After the message is in XML format, the content of the SWIFT document can be validated based on supplied (or customized) rules files. BEA supplies pre-built rules that apply validation rules to tags or groups of tags in the XML document, specific to the SWIFT green books.

After it is parsed and validated, the XML document can be processed according to your site specification. The process flow can end here (for example, parse, convert, validate, and apply business logic). You also can configure all steps on the outbound, which is the reverse of the incoming side.

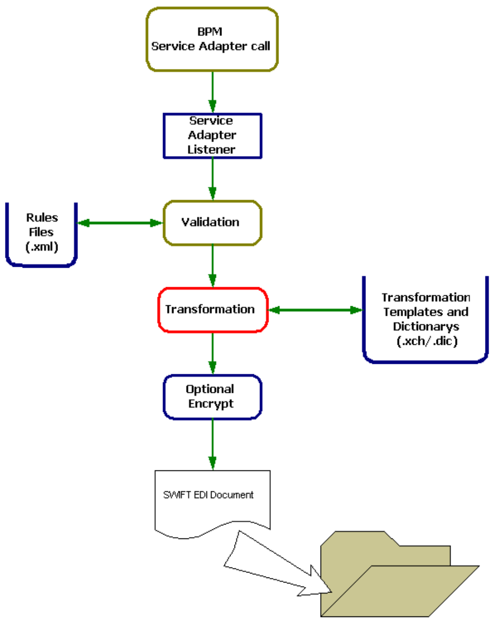
Figure 1-1 Inbound Document Processing



Outbound

The outbound process mirrors (in reverse) the inbound process. A document can be received in XML format and have business logic applied. The rules engine then validates the document and transforms it into a SWIFT document at the pre-emit stage in the process.

Figure 1-2 Outbound Document Processing



Configuring and Validating SWIFT Documents

The BEA WebLogic Adapter for SWIFT comes with a WebLogic JSP-based console that enables authorized users to manage the engine. Listeners, validation rules, and other configuration tasks can be performed here.

Validation occurs against the XML representation of the SWIFT documents. Conversion to XML occurs first. At this stage, the SWIFT dictionaries are used, and structural validation is applied. Further network and content validation of the SWIFT documents using the BEA WebLogic Adapter for SWIFT is performed by a Rules Class, applying rules defined in the rules files. These rules must be created, assigned to the adapter (that is, an alias is created), and applied to the document.

BEA supplies standard rules files that validate a document based on the SWIFT 2001 supplied standards. These files are pre-configured and known to the adapter and also have been related to the documents to be validated. No configuration is required. If customized rules files are used and applied to custom built transformation templates, refer to Appendix A, “Rules System Adapter,” which documents the standard rules and SWIFT-specific rules supplied with the BEA WebLogic Adapter for SWIFT.

The Validation phase may or may not produce errors related to elements in the document tree. After validation, the Acknowledgement Agent processing begins. This phase allows for custom behavior based on the results of the document and its validation results. For SWIFT messages, the default Acknowledgement agent is the XDSWIFTACKAgent. The acknowledgement process is described in detail in Chapter 1, “Acknowledgement Handling.”

2 Creating and Configuring an Event Adapter

An event adapter is the inbound interface from an external protocol. For SWIFT, this may be an MQSeries resource manager (that is, from a Queue Manager and its associated queues), FTP, HTTP, and so forth. This section describes how to create, configure, and test an event adapter and includes the following topics:

- Creating an Application View Folder
- Creating an Event Adapter Application View
- Configuring an Event Adapter Application View
- Testing Views with Application View Console
- Testing Views with WebLogic Integration Studio

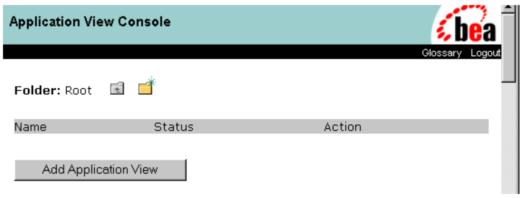
Creating an Application View Folder

Application views reside within WebLogic Integration. WebLogic Integration provides you with a root folder in which you can store all of your application views. If you wish, you can create additional folders to organize related application views into groups.

To create an application view folder:

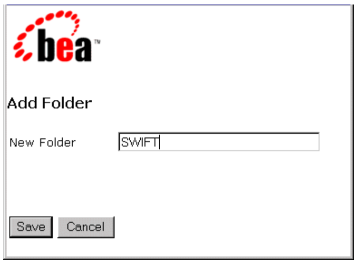
1. Log on to the WebLogic Integration Application View Console at `//appserver-host:port/wlai`; here *`appserver-host`* is the IP address or host name where the WebLogic Integration Server is installed, and *`port`* is the socket on which the server is listening. The port, if not changed during installation, defaults to 7001.

Figure 2-1 Application View Console Main Window



2. Click the new folder icon. The Add Folder window opens:

Figure 2-2 Add Folder Window



New Folder: SWIFT

3. Type a name for the folder and click Save.

You have created the Application View folder.

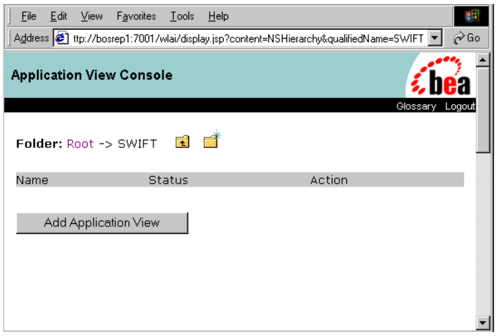
Creating an Event Adapter Application View

To create an event adapter application view:

1. Log on to the WebLogic Integration Application View Console at `//appserver-host:port/wlai`; here `appserver-host` is the IP address or host name where the WebLogic Integration Server is installed, and `port` is the socket on which the server is listening. The port, if not changed during installation, defaults to 7001.

2. At the main page, select the desired application view folder:

Figure 2-3 Application View Console: Selecting the SWIFT Folder

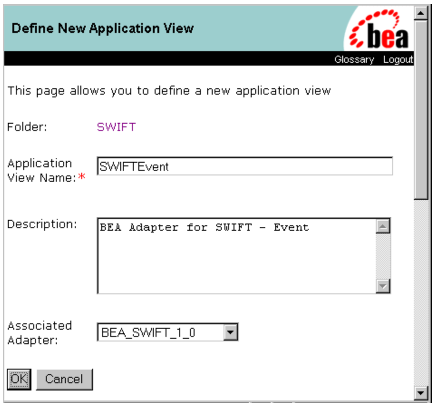


File Edit View Favorites Tools Help
Address ttp://bosrep1:7001/wlai/display.jsp?content=NSHierarchy&qualifiedName=SWIFT
Go
Application View Console
bea
Glossary Logout
Folder: Root -> SWIFT
Name Status Action
Add Application View

3. Click the Add Application View button.

   The Define New Application View page opens.

Figure 2-4 Application View Console: Define New Application View



This page allows you to define a new application view

Folder: SWIFT

Application View Name:* SWIFTEvent

Description: BEA Adapter for SWIFT - Event

Associated Adapter: BEA_SWIFT_1_0

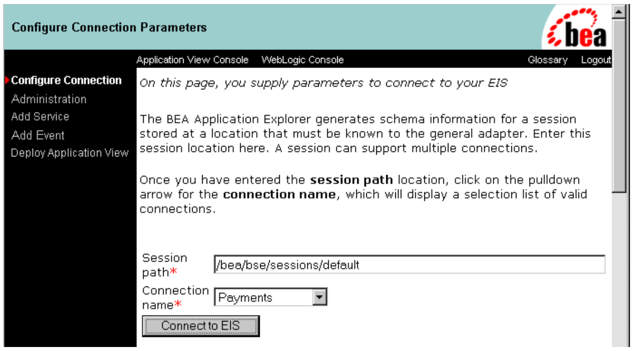
4. In the Application View Name box, type a name (mandatory).

5. In the Description box, type a description for the application view.

6. From the Associated Adapter drop-down list, select `BEA_SWIFT_1_0`.
7. Click OK.

   The Configure Connection Parameters page opens.

Figure 2-5 Configure Connection Parameters Window



On this page, you supply parameters to connect to your EIS

The BEA Application Explorer generates schema information for a session stored at a location that must be known to the general adapter. Enter this session location here. A session can support multiple connections.

Once you have entered the **session path** location, click on the pulldown arrow for the **connection name**, which will display a selection list of valid connections.

Session path*	/bea/bse/sessions/default
Connection name*	Payments

8. In the Session path box, type the name of the BEA WebLogic Adapter for SWIFT session base directory.

   This directory holds your SWIFT schema information and contains the subdirectory `SWIFT/`*`YourConnectionName`*.

   For example, the session base directory might be `/bea/bse/sessions/default`, with the schema repository—containing a repository manifest and schemas—residing in `/bea/bse/sessions/default/SWIFT/Payments`.

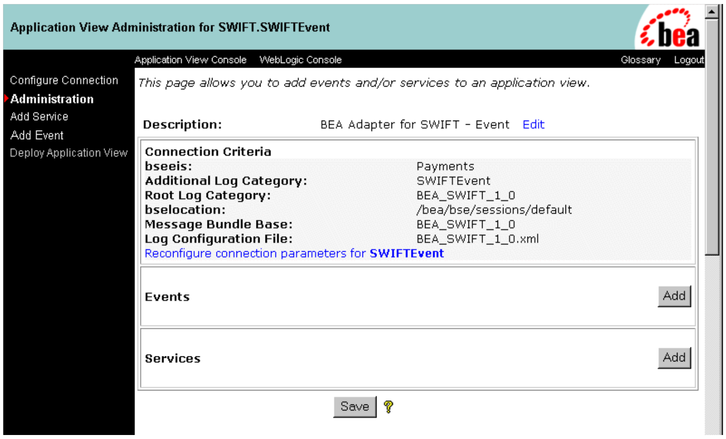
   For more information about schema repositories, see Chapter 5, “Modifying Document Definitions.”

9. From the Connection name drop-down list, select the session name (also known as the connection name).
10. Click Connect to EIS.

Note: You can access the Configure Connection Parameters window (Figure 2-5) when the application view is not deployed, simply by selecting the Reconfigure connection parameters link. If the application view is deployed, you can access the page by first undeploying the application view.

11. When the Application View Administration page opens, click Save.

Figure 2-6 Application View Administration Window



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

Description: BEA Adapter for SWIFT - Event Edit

Connection Criteria

bseeis:	Payments
Additional Log Category:	SWIFTEvent
Root Log Category:	BEA_SWIFT_1_0
bselocation:	/bea/bse/sessions/default
Message Bundle Base:	BEA_SWIFT_1_0
Log Configuration File:	BEA_SWIFT_1_0.xml

Reconfigure connection parameters for SWIFTEvent

Events

Services

You have created the application view for the event adapter.

Note that you must add an event, as described in "Configuring an Event Adapter Application View" on page 2-5, before you can deploy the application view.

Configuring an Event Adapter Application View

An event adapter application view contains all events that are expected to arrive at an instance of the event adapter. You can add several events to an application view. Each event has a schema for the arriving message (A message is also known as a document.) A service should be added for each event that will be used by the application view.

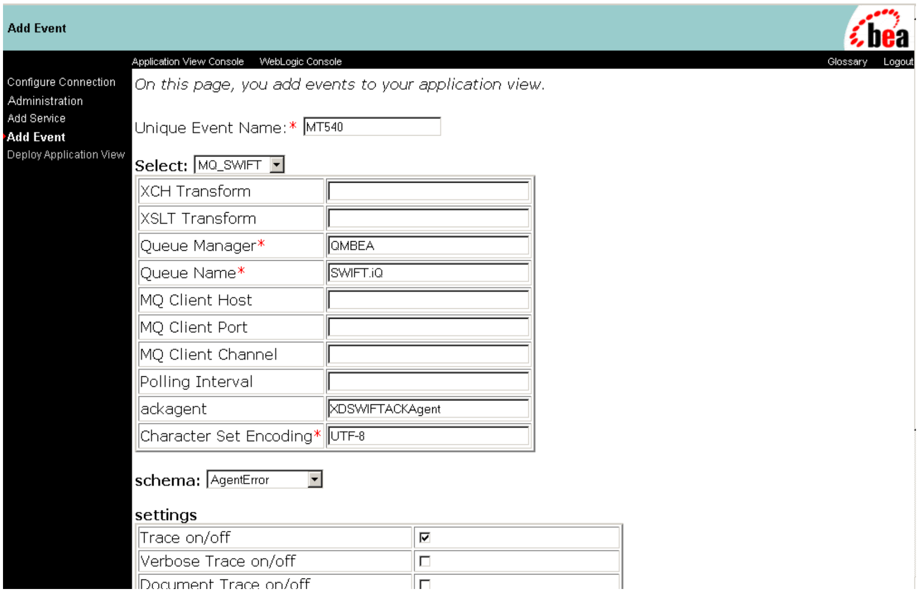
To add an event to an application view and to deploy an event adapter:

1. Log on to the WebLogic Integration Application View Console at `//appserver-host:port/wlai`.
2. Select the folder in which this application view resides and then select the application view.

3. In the Administration page of the WebLogic Integration Application View Console, select Add Event.

   The Add Event page opens.

Figure 2-7 Application View Console: Add Event - MQSeries



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

Unique Event Name:* MT540

Select: MQ_SWIFT

XCH Transform	
XSLT Transform	
Queue Manager*	QMBEA
Queue Name*	SWIFT.iQ
MQ Client Host	
MQ Client Port	
MQ Client Channel	
Polling Interval	
ackagent	XDSWIFTACKAgent
Character Set Encoding*	UTF-8

schema: AgentError

settings

Trace on/off	✓
Verbose Trace on/off	☐
Document Trace on/off	☐

The BEA WebLogic Adapter for SWIFT supports multiple protocols for receiving its SWIFT messages.

4. In the Listener drop-down list box, select the desired transport protocol.

   The screen displays the associated parameters for the chosen listener. The properties in Figure 2-7 correspond to the MQSeries communication and transformation settings that the event adapter will use.

   The schema drop-down list corresponds to the list of events in the schema repository.

The following table describes the communication settings for the MQSeries transport protocol.

Table 2-1 Event Properties MQ Series

Property	Description	Type	Sample Value	Element
ackagent	The agent responsible for processing acknowledgement or non-acknowledgment of ISO documents. The default acknowledgment agent generates an empty acknowledgment document or lists all failed validation rules in the non-acknowledgment document.			
Character Set Encoding	Sets the character set encoding to be used.	string		
MQ Client Channel	For MQ Client only. Channel between an MQ Client and MQ Server.	string		<channel>
MQ Client Host	For MQ Client only. Host on which MQ Server is located.	string		<host>
MQ Client Port	For MQ Client only. Port number to connect to an MQ Server.	integer		<port>
Polling Interval	Indicates the frequency in seconds that the event returns control to the WebLogic Server to determine if a stop or recycle of the event has been requested. The event is constantly connected to the queue to retrieve incoming messages. The default value is 2 seconds.	duration		<timeout>
Queue Manager	Name of the MQSeries Queue Manager to be used.	string	OMBEA	<manager>

Table 2-1 Event Properties MQ Series

Property	Description	Type	Sample Value	Element
Request Queue Name	Queue where request documents are received.	string	TEST.iO	<queue>
XCH Transform	Name of the `.xch` transform template file used by the XCH transform type in the XML to XML transformation phase. For more information, see Chapter 4, "Transforming Document Formats."	string		<in_xmlg>
XSLT Transform	Name of the XSLT stylesheet used by the XSLT transform type in the XML to XML transformation phase. **Note:** Do not enter an `.mfl` extension for a Message Format Language (MFL) file; these files are not stored with an extension. For more information, see Chapter 4, "Transforming Document Formats."	string		<in_xslt>

5. Choose from the following trace settings for event properties.

Table 2-2 Trace Settings

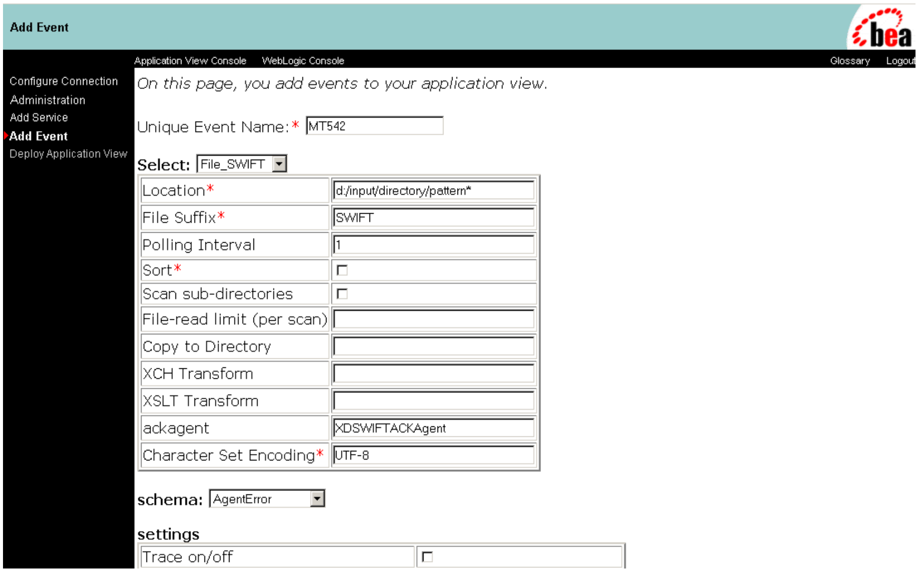
Property	Description
Trace on/off	Generates a basic trace that displays the input XML (up to 300 bytes) before parsing, and shows the request being processed. For more information about tracing, see Chapter 7, *How Adapters Use Tracing*.

Table 2-2 Trace Settings

Property	Description
Verbose Trace on/off	Generates a trace that displays configuration parameters used by the adapter. For more information about tracing, see Chapter 7, *How Adapters Use Tracing*.
Document Trace on/off	Generates a trace that displays the input document after it was analyzed and the response document being returned. For more information about tracing, see Chapter 7, *How Adapters Use Tracing*.

For the File adapter, the following screen shows the applicable parameters:

Figure 2-8 Application View Console: Add Event - File



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

Unique Event Name:* MT542

Select: File_SWIFT

Location*	d:/input/directory/pattern*
File Suffix*	SWIFT
Polling Interval	1
Sort*	☐
Scan sub-directories	☐
File-read limit (per scan)	
Copy to Directory	
XCH Transform	
XSLT Transform	
ackagent	XDSWIFTACKAgent
Character Set Encoding*	UTF-8

schema: AgentError

settings

Trace on/off	☐

The following table describes the properties for File.

Table 2-3 Event Properties - File

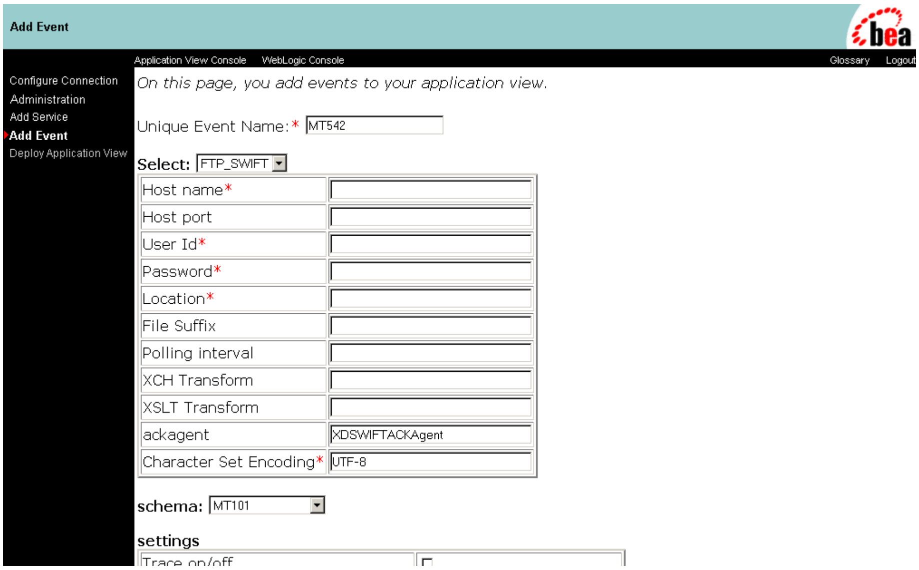
Setting	Meaning/Properties
ackagent	**Type/Value:** String **Description:** The agent responsible for processing acknowledgement or non-acknowledgment of ISO documents. The default acknowledgment agent generates an empty acknowledgment document or lists all failed validation rules in the non-acknowledgment document.
Character set encoding (*Required)	**Type/Value:** String **Description:** Document character set.
Copy to Directory	**Type/Value:** String **Description:** Directory to which the document being processed will be copied. This allows you to keep copies of the documents.
File Suffix (*Required)	**Type/Value:** String **Description:** Mandatory suffix of the input file.
File-read limit (per scan)	**Type/Value:** Integer **Description:** The number of files read per sweep of the File directory location.
Location (*Required)	**Type/Value:** String **Description:** The location and name of the file pattern for the SWIFT document.
Polling Interval	**Type/Value:** Duration **Description:** Indicates the frequency in seconds that the event returns control to the WebLogic Server to determine if a stop or recycle of the event has been requested. The event is constantly connected to the queue to retrieve incoming messages. The default value is 2 seconds

Table 2-3 Event Properties - File

Setting	Meaning/Properties
Scan sub-directories	**Type/Value:** Boolean **Description:** Flag to indicate whether files should be read from subdirectories.
Sort	**Type/Value:** Boolean **Description:** Flag to indicate whether the list of files pulled from the location should be sorted for processing order.
XCH Transform	**Type/Value:** String **Description:** Name of the `.xch` transform template file used by the XCH transform type in the XML to XML transformation phase. For more information, see Chapter 4, "Transforming Document Formats."
XSLT Transform	**Type/Value:** String **Description:** Name of the XSLT stylesheet used by the XSLT transform type in the XML to XML transformation phase. **Note:** Do not enter an `.mfl` extension for a Message Format Language (MFL) file; these files are not stored with an extension. For more information, see Chapter 4, "Transforming Document Formats."

For the FTP adapter, the following screen shows the applicable parameters.

Figure 2-9 Application View Console: Add Event - FTP



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

Unique Event Name:* MT542

Select: FTP_SWIFT

Host name*	
Host port	
User Id*	
Password*	
Location*	
File Suffix	
Polling interval	
XCH Transform	
XSLT Transform	
ackagent	XDSWIFTACKAgent
Character Set Encoding*	UTF-8

schema: MT101

settings

Trace on/off

For the TCP adapter, the following screen shows the applicable parameters.

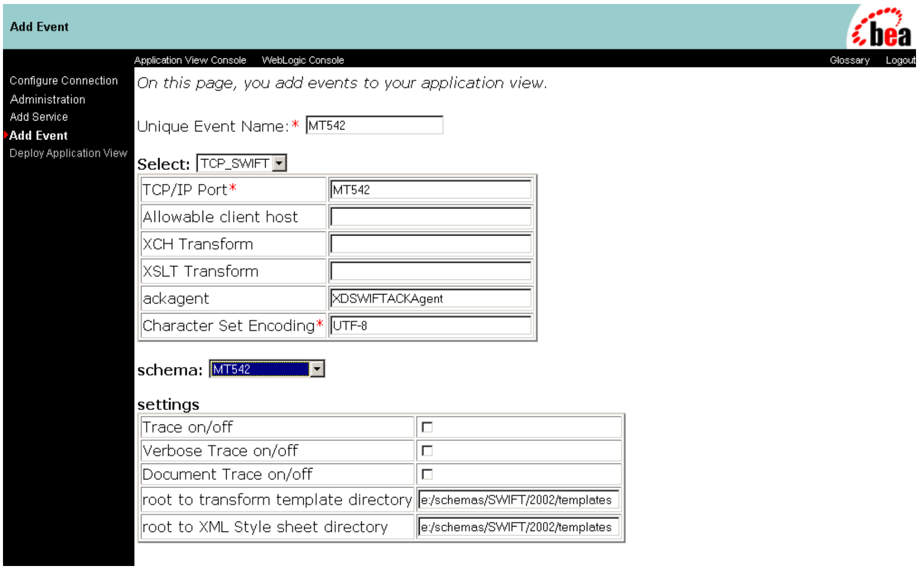
Table 2-4 Event Properties - FTP

Setting	Meaning/Properties
ackagent	The agent responsible for processing acknowledgement or non-acknowledgment of ISO documents. The default acknowledgment agent generates an empty acknowledgment document or lists all failed validation rules in the non-acknowledgment document.
Character Set Encoding	Sets the character set encoding to be used.
File Suffix	**Type/Value:** String **Description:** Suffix of source files.
Host name* (*Required)	**Type/Value:** String **Description:** FTP target system.
Host port	Alternate port number.

Table 2-4 Event Properties - FTP (Continued)

Setting	Meaning/Properties
Location*	**Type/Value:** String **Description:** Location on FTP server for source files.
Password* (*Required)	**Type/Value:** String **Description:** Password for user account to use when connecting to protocol host.
Polling Interval	Indicates the frequency in seconds that the event returns control to the WebLogic Server to determine if a stop or recycle of the event has been requested. The event is constantly connected to the queue to retrieve incoming messages. The default value is 2 seconds.
User ID* (*Required)	**Type/Value:** String **Description:** User account ID to use when connecting to protocol host.
XCH Transform	Name of the `.xch` transform template file used by the XCH transform type in the XML to XML transformation phase. For more information, see Chapter 4, "Transforming Document Formats."
XSLT Transform	Name of the XSLT stylesheet used by the XSLT transform type in the XML to XML transformation phase. **Note:** Do not enter an `.mfl` extension for a Message Format Language (MFL) file; these files are not stored with an extension. For more information, see Chapter 4, "Transforming Document Formats."

Figure 2-10 Application View Console: Add Event - TCP



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

Unique Event Name:* MT542

Select: TCP_SWIFT

TCP/IP Port*	MT542
Allowable client host	
XCH Transform	
XSLT Transform	
ackagent	XDSWIFTACKAgent
Character Set Encoding*	UTF-8

schema: MT542

settings

Trace on/off	☐
Verbose Trace on/off	☐
Document Trace on/off	☐
root to transform template directory	e:/schemas/SWIFT/2002/templates
root to XML Style sheet directory	e:/schemas/SWIFT/2002/templates

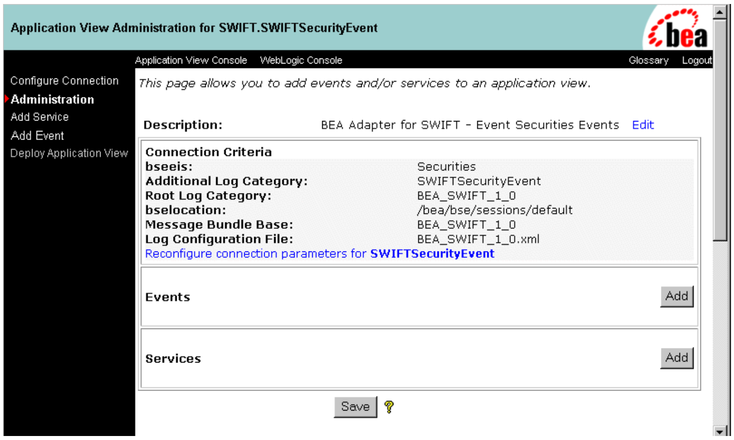
Table 2-5 Event Properties - TCP

Setting	Meaning/Properties
ackagent	The agent responsible for processing acknowledgement or non-acknowledgment of ISO documents. The default acknowledgment agent generates an empty acknowledgment document or lists all failed validation rules in the non-acknowledgment document.
Allowable Client Host	**Type/Value:** String **Description:** Host name or host address of client restricted to accessing this event adapter.
Character set encoding (*Required)	**Type/Value:** String **Description:** Document character set.
TCP Port* (*Required)	**Type/Value:** Integer **Description:** TCP listening port.
XCH Transform	Name of the `.xch` transform template file used by the XCH transform type in the XML to XML transformation phase. For more information, see Chapter 4, "Transforming Document Formats."

Table 2-5 Event Properties - TCP

XSLT Transform	Name of the XSLT stylesheet used by the XSLT transform type in the XML to XML transformation phase. **Note:** Do not enter an `.mfl` extension for a Message Format Language (MFL) file; these files are not stored with an extension. For more information, see Chapter 4, "Transforming Document Formats."

Figure 2-11 Application View Administration Window



Application View Administration for SWIFT.SWIFTSecurityEvent

Application View Console WebLogic Console Glossary Logout

Configure Connection
Administration
Add Service
Add Event
Deploy Application View

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

Description: BEA Adapter for SWIFT - Event Securities Events Edit

Connection Criteria

bseeis:	Securities
Additional Log Category:	SWIFTSecurityEvent
Root Log Category:	BEA_SWIFT_1_0
bselocation:	/bea/bse/sessions/default
Message Bundle Base:	BEA_SWIFT_1_0
Log Configuration File:	BEA_SWIFT_1_0.xml

Reconfigure connection parameters for **SWIFTSecurityEvent**

Events Add

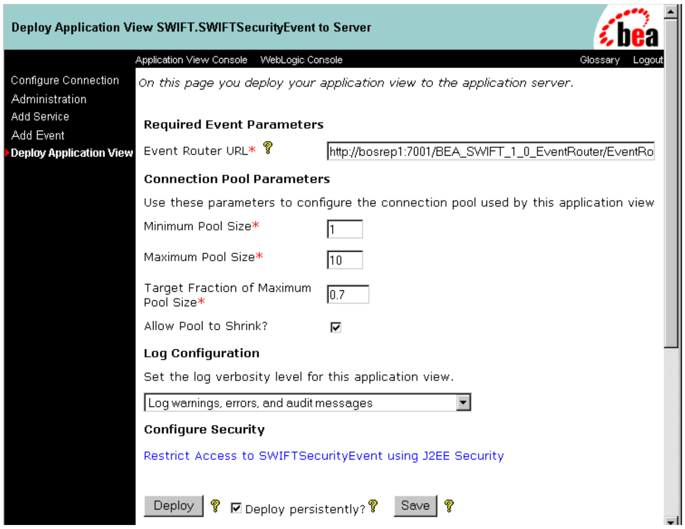
Services Add

Save

6. Click Add.
7. When the next screen opens, click Continue.

   The Deploy Application View page opens.

Figure 2-12 Deploy Application View Window



Deploy Application View SWIFT.SWIFTSecurityEvent to Server

On this page you deploy your application view to the application server.

Required Event Parameters

Event Router URL* http://bosrep1:7001/BEA_SWIFT_1_0_EventRouter/EventRo

Connection Pool Parameters

Use these parameters to configure the connection pool used by this application view

Minimum Pool Size*	1
Maximum Pool Size*	10
Target Fraction of Maximum Pool Size*	0.7
Allow Pool to Shrink?	✓

Log Configuration

Set the log verbosity level for this application view.

Log warnings, errors, and audit messages

Configure Security

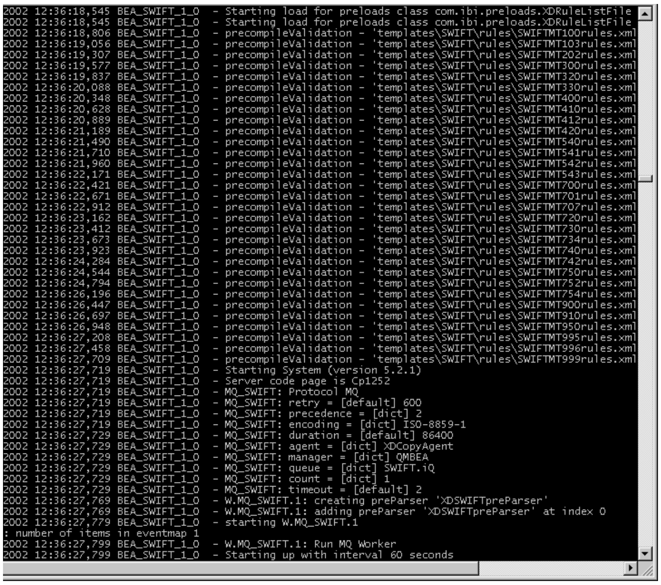
Restrict Access to SWIFTSecurityEvent using J2EE Security

8. Modify event parameters, connection pool parameters, log configuration, and security as necessary.

9. Click Deploy to save and deploy the event adapter.

   In the WebLogic Server Log, you should see the following entries as the event adapter deploys.

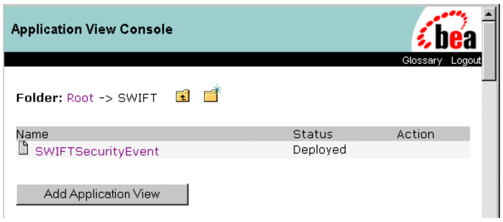
Figure 2-13 WebLogic Server Log



```
2002 12:36:18,545 BEA_SWIFT_1_0  - Starting load for preloads class com.ibi.preloads.XDRuleListFile
2002 12:36:18,545 BEA_SWIFT_1_0  - Starting load for preloads class com.ibi.preloads.XDRuleListFile
2002 12:36:18,806 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT100rules.xml
2002 12:36:19,056 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT103rules.xml
2002 12:36:19,307 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT202rules.xml
2002 12:36:19,577 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT300rules.xml
2002 12:36:19,837 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT320rules.xml
2002 12:36:20,088 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT330rules.xml
2002 12:36:20,348 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT400rules.xml
2002 12:36:20,628 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT410rules.xml
2002 12:36:20,889 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT412rules.xml
2002 12:36:21,189 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT420rules.xml
2002 12:36:21,490 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT540rules.xml
2002 12:36:21,710 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT541rules.xml
2002 12:36:21,960 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT542rules.xml
2002 12:36:22,171 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT543rules.xml
2002 12:36:22,421 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT700rules.xml
2002 12:36:22,671 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT701rules.xml
2002 12:36:22,912 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT707rules.xml
2002 12:36:23,162 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT720rules.xml
2002 12:36:23,412 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT730rules.xml
2002 12:36:23,673 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT734rules.xml
2002 12:36:23,923 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT740rules.xml
2002 12:36:24,284 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT742rules.xml
2002 12:36:24,544 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT750rules.xml
2002 12:36:24,794 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT752rules.xml
2002 12:36:26,196 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT754rules.xml
2002 12:36:26,447 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT900rules.xml
2002 12:36:26,697 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT910rules.xml
2002 12:36:26,948 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT950rules.xml
2002 12:36:27,208 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT995rules.xml
2002 12:36:27,458 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT996rules.xml
2002 12:36:27,709 BEA_SWIFT_1_0  - precompileValidation - 'templates\SWIFT\rules\SWIFTMT999rules.xml
2002 12:36:27,719 BEA_SWIFT_1_0  - Starting System (version 5.2.1)
2002 12:36:27,719 BEA_SWIFT_1_0  - Server code page is Cp1252
2002 12:36:27,719 BEA_SWIFT_1_0  - MQ_SWIFT: Protocol MQ
2002 12:36:27,719 BEA_SWIFT_1_0  - MQ_SWIFT: retry = [default] 600
2002 12:36:27,719 BEA_SWIFT_1_0  - MQ_SWIFT: precedence = [dict] 2
2002 12:36:27,719 BEA_SWIFT_1_0  - MQ_SWIFT: encoding = [dict] ISO-8859-1
2002 12:36:27,729 BEA_SWIFT_1_0  - MQ_SWIFT: duration = [default] 86400
2002 12:36:27,729 BEA_SWIFT_1_0  - MQ_SWIFT: agent = [dict] XDCopyAgent
2002 12:36:27,729 BEA_SWIFT_1_0  - MQ_SWIFT: manager = [dict] QMBEA
2002 12:36:27,729 BEA_SWIFT_1_0  - MQ_SWIFT: queue = [dict] SWIFT.iQ
2002 12:36:27,729 BEA_SWIFT_1_0  - MQ_SWIFT: count = [dict] 1
2002 12:36:27,729 BEA_SWIFT_1_0  - MQ_SWIFT: timeout = [default] 2
2002 12:36:27,769 BEA_SWIFT_1_0  - W.MQ_SWIFT.1: creating preParser 'XDSWIFTpreParser'
2002 12:36:27,769 BEA_SWIFT_1_0  - W.MQ_SWIFT.1: adding preParser 'XDSWIFTpreParser' at index 0
2002 12:36:27,779 BEA_SWIFT_1_0  - starting W.MQ_SWIFT.1
: number of items in eventmap 1
2002 12:36:27,799 BEA_SWIFT_1_0  - W.MQ_SWIFT.1: Run MQ Worker
2002 12:36:27,799 BEA_SWIFT_1_0  - Starting up with interval 60 seconds
```

10. To validate that the application view was successfully deployed, go to the main Application View Console page and select the folder where you created the application view. You should see the name of the new application view.

Figure 2-14 Application View Console: New Application View



Folder: Root -> SWIFT

Name	Status	Action
SWIFTSecurityEvent	Deployed	

You have finished configuring the event adapter application view.

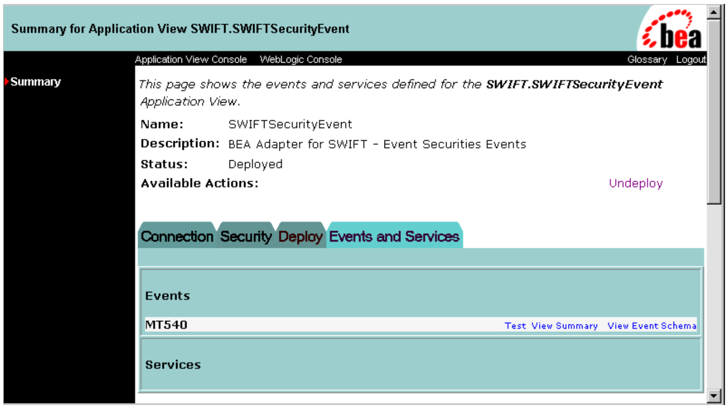
You can confirm that you configured it correctly and that it can successfully receive events. Follow the instructions in "Testing Views with Application View Console" on page 2-18 and "Testing Views with WebLogic Integration Studio" on page 2-21.

Testing Views with Application View Console

To confirm that a deployed event adapter application view is correctly configured and can receive events:

1. Log on to the WebLogic Integration Application View Console at `//`*`appserver-host:port`*`/wlai`.
2. Select the folder in which the application view resides and then select the application view. The Summary page opens:

Figure 2-15 Application View Console Summary Window


Summary for Application View SWIFT.SWIFTSecurityEvent

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

Name: SWIFTSecurityEvent
Description: BEA Adapter for SWIFT - Event Securities Events
Status: Deployed
Available Actions: Undeploy

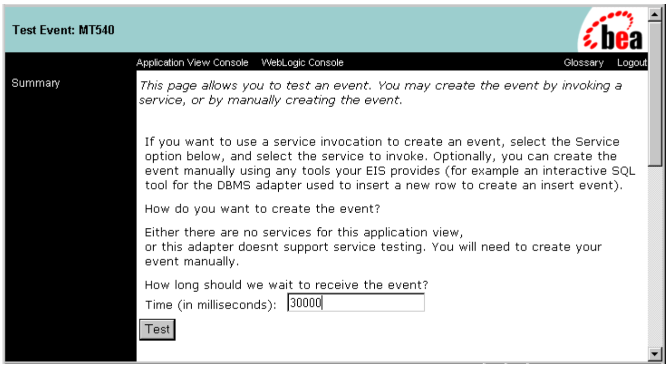
Events
MT540 Test View Summary View Event Schema

Services

3. Click Test for one of the application view's events.

   The Test Event page opens:

Figure 2-16 Application View Console: Test Event Window



Test Event: MT540

Application View Console WebLogic Console Glossary Logout

Summary

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, and select the service to invoke. Optionally, you can create the event manually using any tools your EIS provides (for example an interactive SQL tool for the DBMS adapter used to insert a new row to create an insert event).

How do you want to create the event?

Either there are no services for this application view, or this adapter doesnt support service testing. You will need to create your event manually.

How long should we wait to receive the event?

Time (in milliseconds): 30000

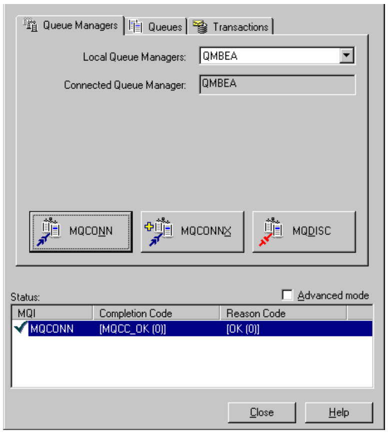
Test

4. Enter 3000 (or a higher value) in the Time box. This provides a 30-second period during which, in the following step, you can access the IBM MQSeries API Exerciser (or your favorite utility) to manually invoke a request from MQSeries to your event adapter.

5. Click Test.

6. Open the MQSeries API Exerciser and select the Queue Manager that you assigned to the event adapter in the Add Event page of the Application View Console.

Figure 2-17 MQSeries API Exerciser: Queue Managers Tab



Local Queue Managers: QMBEA

Connected Queue Manager: QMBEA

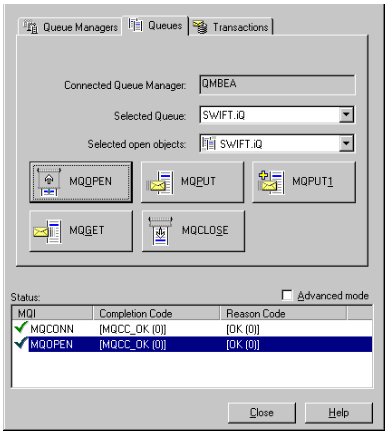
Status:

MQI	Completion Code	Reason Code
✓ MQCONN	[MQCC_OK (0)]	[OK (0)]

7. Click MQCONN.

8. On the Queues tab, in the Selected Queue drop-down list, select the queue that you had assigned to the event adapter in the Add Event page of the Application View Console.

Figure 2-18 MQSeries API Exerciser: Queues Tab



Connected Queue Manager: QMBEA
Selected Queue: SWIFT.iQ
Selected open objects: SWIFT.iQ

Status: ☐ Advanced mode

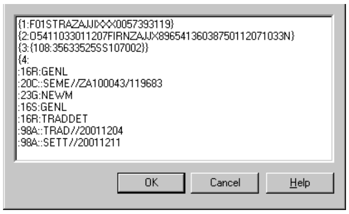
MQI	Completion Code	Reason Code
✓ MQCONN	[MQCC_OK (0)]	[OK (0)]
✓ MQOPEN	[MQCC_OK (0)]	[OK (0)]

9. Click MQOPEN.

10. Click MQPUT.

    The MQPUT dialog box appears.

Figure 2-19 MQPUT Window with Sample

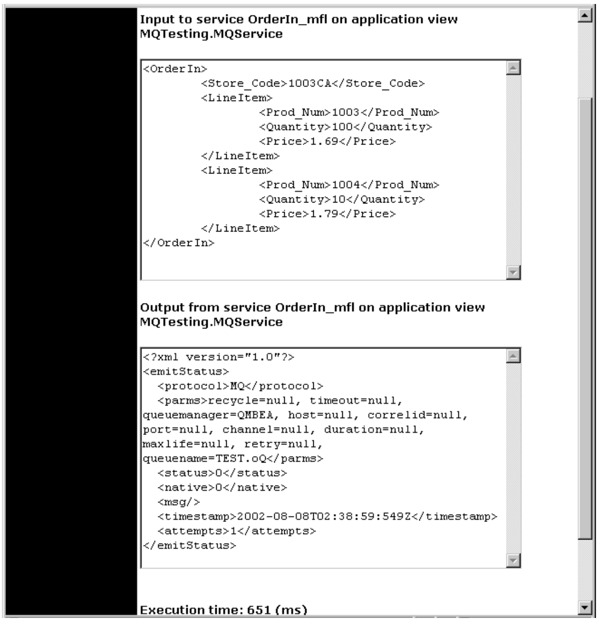


```
{1:F01STRAZAJJIXXX0057393119}
{2:O5411033011207FIRNZAJJX89654136038750112071033N}
{3:{108:35633525SS107002}}
{4:
:16R:GENL
:20C::SEME//ZA100043/119683
:23G:NEWM
:16S:GENL
:16R:TRADDET
:98A::TRAD//20011204
:98A::SETT//20011211
```

11. Copy and paste a sample instance document that matches the schema for the event that you are testing:

    The Test Result page displays the event's result.

Figure 2-20 Application View Console: Test Result



Input to service OrderIn_mfl on application view MQTesting.MQService

```
<OrderIn>
        <Store_Code>1003CA</Store_Code>
        <LineItem>
                <Prod_Num>1003</Prod_Num>
                <Quantity>100</Quantity>
                <Price>1.69</Price>
        </LineItem>
        <LineItem>
                <Prod_Num>1004</Prod_Num>
                <Quantity>10</Quantity>
                <Price>1.79</Price>
        </LineItem>
</OrderIn>
```

Output from service OrderIn_mfl on application view MQTesting.MQService

```
<?xml version="1.0"?>
<emitStatus>
  <protocol>MQ</protocol>
  <parms>recycle=null, timeout=null,
queuemanager=QMBEA, host=null, correlid=null,
port=null, channel=null, duration=null,
maxlife=null, retry=null,
queuename=TEST.oQ</parms>
  <status>0</status>
  <native>0</native>
  <msg/>
  <timestamp>2002-08-08T02:38:59:549Z</timestamp>
  <attempts>1</attempts>
</emitStatus>
```

Execution time: 651 (ms)

If you wait longer than a minute and do not receive the event's result, you should assume that there is a problem with the event adapter application view. You can examine the WebLogic Server Log for information about the event's activity.

Otherwise, you have now confirmed that the event adapter application view is correctly configured and can receive events.

Testing Views with WebLogic Integration Studio

To confirm that a deployed event adapter application view is correctly configured and can receive events:

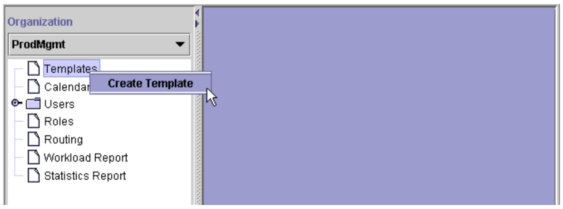
1. Start the WebLogic Integration Studio:

On a Windows system, choose Start→Programs→BEA WebLogic Platform 7.0→WebLogic Integration 7.0→Studio.

On a UNIX system, go to the WLI_HOME/bin directory and run the studio command.

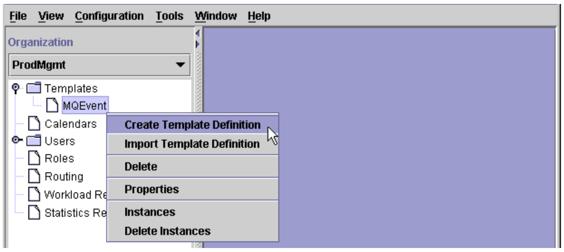
2. Log on to the WebLogic Integration Studio.
3. In the Organization pane, select an organization to create a new business process management workflow template.
4. Right-click Templates and select Create Template:

Figure 2-21 WebLogic Integration Studio: Create Template



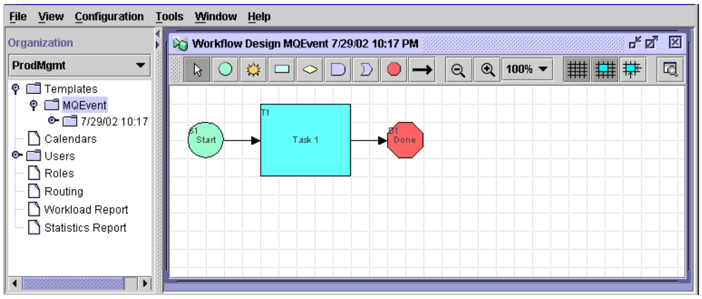
5. Right-click the new template and select Create Template Definition:

Figure 2-22 WebLogic Integration Studio: Create Template Definition



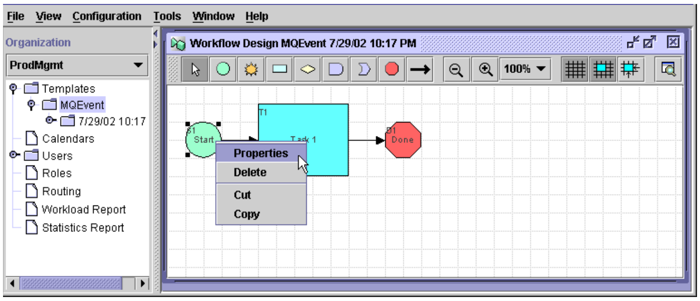
The template appears in WebLogic Integration Studio:

Figure 2-23 WebLogic Integration Studio: New Template



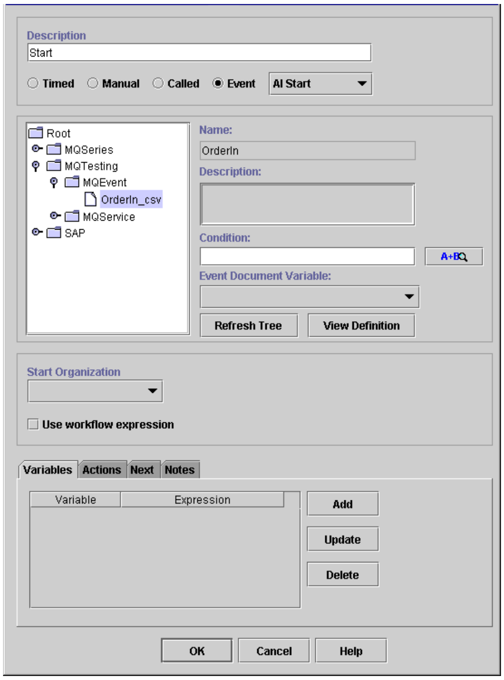
6. Right-click the Start node and select Properties:

Figure 2-24 WebLogic Integration Studio: Start Node Shortcut Menu



The Start Properties dialog box appears.

Figure 2-25 WebLogic Integration Studio: Start Properties


Description: Start

( ) Timed ( ) Manual ( ) Called (●) Event — AI Start

- Root
  - MQSeries
  - MQTesting
    - MQEvent
      - OrderIn_csv
    - MQService
  - SAP

Name: OrderIn

Description:

Condition:

Event Document Variable:

Start Organization:

☐ Use workflow expression

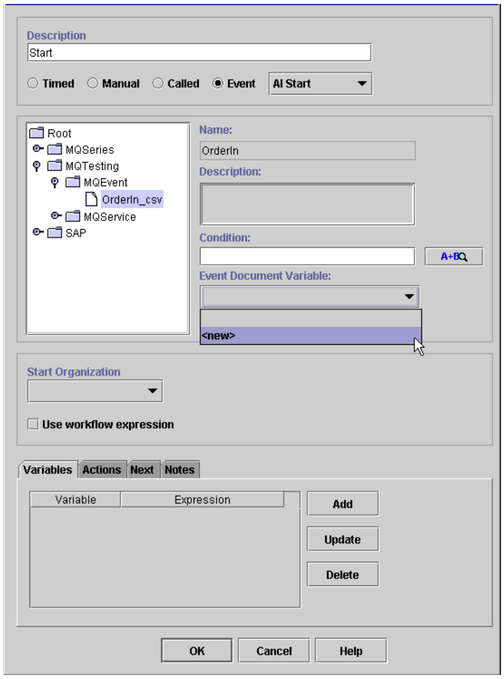
7. Click Event and then, AI Start.

8. In the event explorer, browse the application view folders and select the application view that corresponds to the event adapter.

9. Open the event adapter and select the desired event.

10. Select <new> from the Event Document Variable drop-down list.

Figure 2-26 Start Properties - New Document Variable



Description: Start

( ) Timed ( ) Manual ( ) Called (●) Event — AI Start

Root
- MQSeries
- MQTesting
  - MQEvent
    - OrderIn_csv
  - MQService
- SAP

Name: OrderIn

Description:

Condition:

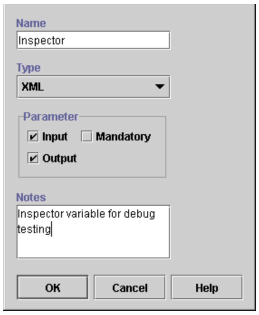
Event Document Variable: <new>

Start Organization

☐ Use workflow expression

The Variable Properties dialog box appears.

Figure 2-27 Variable Properties Dialog Box



Name: Inspector

Type: XML

Parameter: ☑ Input ☐ Mandatory ☑ Output

Notes: Inspector variable for debug testing

11. Type a name for the new variable.

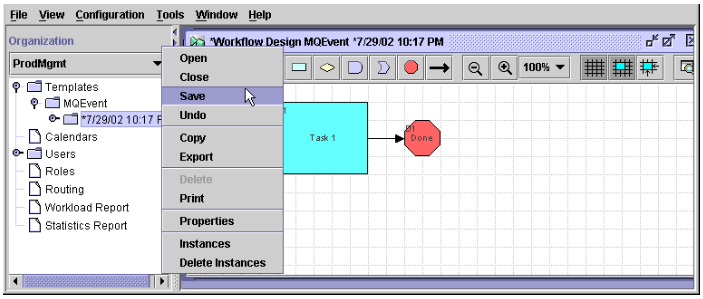
12. Select the variable type, XML.

13. Check the Input and Output options in the Parameter group.

14. Click OK.

15. Right-click the template in the left pane of WebLogic Integration Studio and select Save from the shortcut menu.

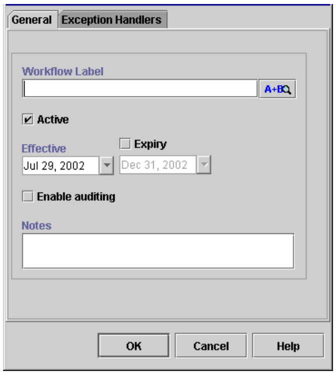
Figure 2-28 WebLogic Integration Studio: Save Template



16. Right-click the event definition folder and select Properties.

    The Template Definition dialog box appears.

Figure 2-29 Template Definition Dialog Box



Workflow Label:
[✓] Active
Effective: Jul 29, 2002
[ ] Expiry: Dec 31, 2002
[ ] Enable auditing
Notes:

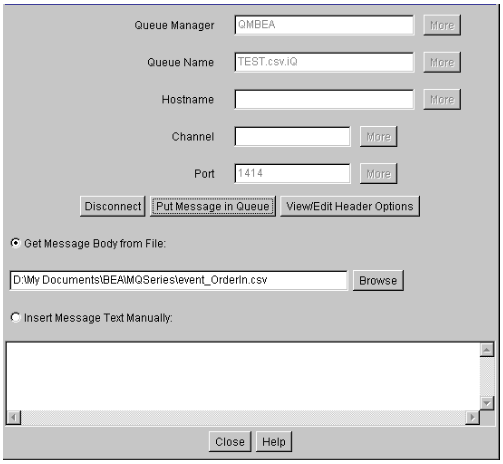
17. Ensure that Active is checked and click OK.

    You may now initiate events from your Enterprise Information System. For the BEA WebLogic Adapter for SWIFT, you can create events through a business application or through an MQSeries utility.

    This utility assists in writing messages into the queue.

    For example, you could use the PutMessage utility in MQSeries SupportPac MA0J to put messages into a queue to which the event adapter is configured:

Figure 2-30 MQSeries PutMessage


Queue Manager: QMBEA

Queue Name: TEST.csv.iQ

Hostname:

Channel:

Port: 1414

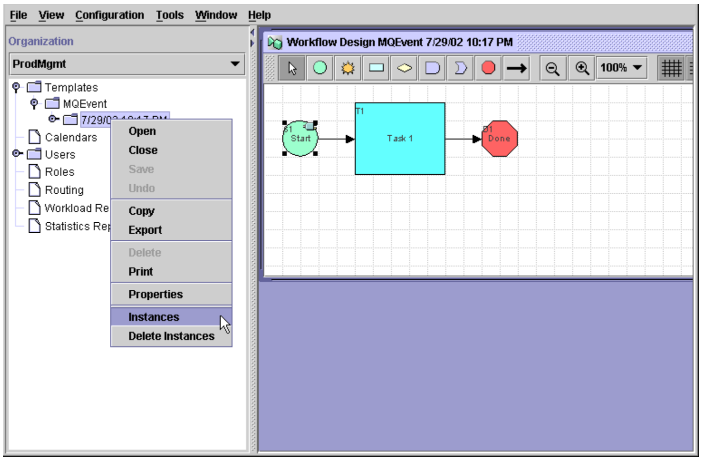
(●) Get Message Body from File:

D:\My Documents\BEA\MQSeries\event_OrderIn.csv

( ) Insert Message Text Manually:

1. Enter free-form text or source the data from a file.
2. Then specify a Queue Manager, queue name, and port for your locally accessible MQ installation.
3. Select the message or message source file and click Put Message in Queue.
4. Return to WebLogic Integration Studio.
5. Right-click the event definition folder and select Instances.

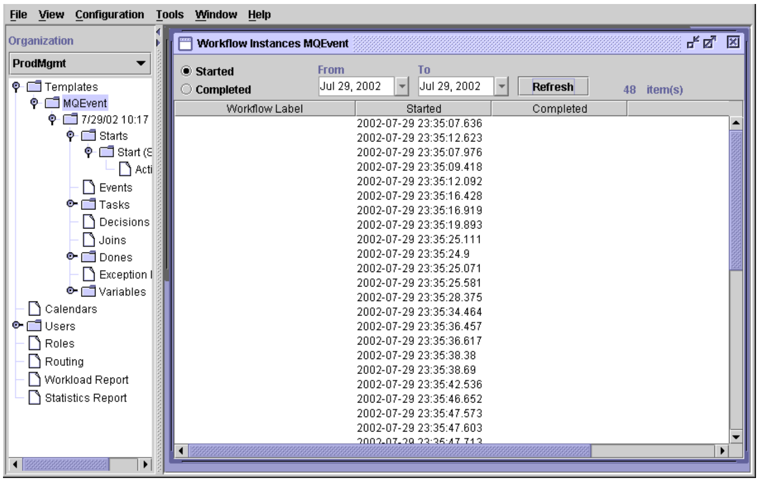
Figure 2-31 WebLogic Integration Studio: Event Definition Folder



The Workflow Instances for your event definition appear.

You can now track execution of your workflow.

Figure 2-32 WebLogic Integration Studio: Workflow Instances

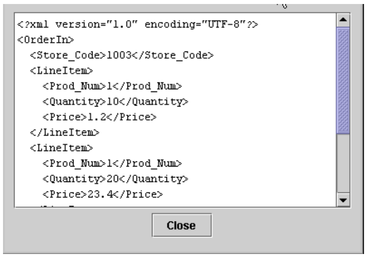


Workflow Label	Started	Completed
	2002-07-29 23:35:07.636	
	2002-07-29 23:35:12.623	
	2002-07-29 23:35:07.976	
	2002-07-29 23:35:09.418	
	2002-07-29 23:35:12.092	
	2002-07-29 23:35:16.428	
	2002-07-29 23:35:16.919	
	2002-07-29 23:35:19.893	
	2002-07-29 23:35:25.111	
	2002-07-29 23:35:24.9	
	2002-07-29 23:35:25.071	
	2002-07-29 23:35:25.581	
	2002-07-29 23:35:28.375	
	2002-07-29 23:35:34.464	
	2002-07-29 23:35:36.457	
	2002-07-29 23:35:36.617	
	2002-07-29 23:35:38.38	
	2002-07-29 23:35:38.69	
	2002-07-29 23:35:42.536	
	2002-07-29 23:35:46.652	
	2002-07-29 23:35:47.573	
	2002-07-29 23:35:47.603	

6. Click Started and click Refresh. You should see a list of started work flows:
7. Right-click any instance of the workflow and select Variables.

   The Workflow Variables dialog box appears.
8. Click View XML to see the entire contents of the workflow message or document:

Figure 2-33 WebLogic Integration Studio: Workflow Variables



```
<?xml version="1.0" encoding="UTF-8"?>
<OrderIn>
  <Store_Code>1003</Store_Code>
  <LineItem>
    <Prod_Num>1</Prod_Num>
    <Quantity>10</Quantity>
    <Price>1.2</Price>
  </LineItem>
  <LineItem>
    <Prod_Num>1</Prod_Num>
    <Quantity>20</Quantity>
    <Price>23.4</Price>
```

3 Creating and Configuring a Service Adapter

A service adapter for SWIFT is WebLogic Integration's interface to SWIFT. It enables your business processes to write output in SWIFT format. This section contains the following topics:

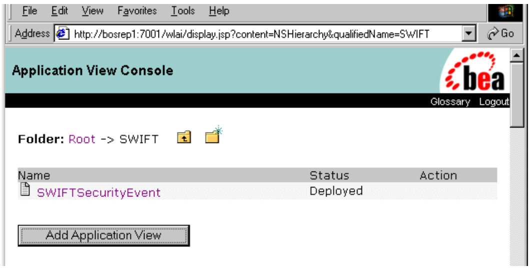
- Creating Service Adapter Application Views
- Configuring Service Adapter Application Views
- Testing SWIFT Service Adapters

Creating Service Adapter Application Views

A service adapter for SWIFT is configured for all services that must send data to SWIFT.

1. Click the Add Application View button on the Application View Console.

Figure 3-1 Application View Console Window



http://bosrep1:7001/wlai/display.jsp?content=NSHierarchy&qualifiedName=SWIFT

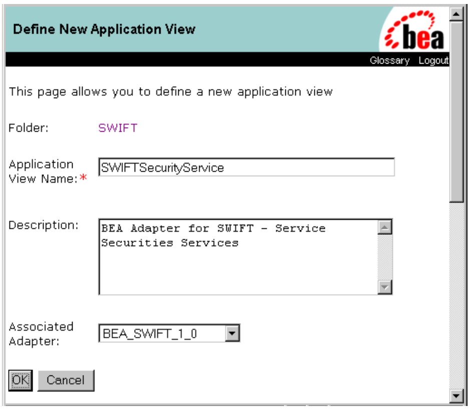
Application View Console

Folder: Root -> SWIFT

Name	Status	Action
SWIFTSecurityEvent	Deployed	

The Define New Application View page opens.

Figure 3-2 Define New Application View



Define New Application View

This page allows you to define a new application view

Folder: SWIFT

Application View Name:* SWIFTSecurityService

Description: BEA Adapter for SWIFT - Service Securities Services

Associated Adapter: BEA_SWIFT_1_0

2. In the Application View Name box, type a name (mandatory).

3. In the Description box, type a description for the application view.

4. From the Associated Adapter drop-down list, select BEA_SWIFT_1_0.

5. Click OK.

6. Click OK. The Configure Connection Parameters page opens.

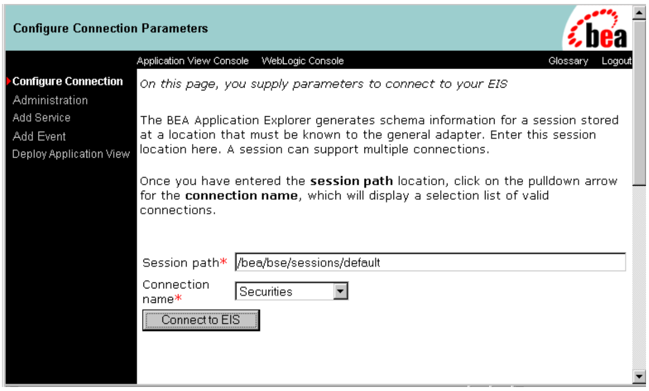
7. Type the name of the BEA WebLogic Adapter for SWIFT session base directory in the Session path box. This directory holds your SWIFT schema information, and contains the subdirectory `SWIFT/`*`YourConnectionName`*.

   For example, the session base directory might be `d:\bea\bse\sessions\default`, with the schema repository—containing a repository manifest and schemas—residing in

`/bea/bse/sessions/default/SWIFT/QMBEA`. For more information about schema repositories, see Chapter 5, "Modifying Document Definitions."

8. Select the session name—also known as the connection name—from the Connection name drop-down list.

Figure 3-3 Configure Connection Parameters



On this page, you supply parameters to connect to your EIS

The BEA Application Explorer generates schema information for a session stored at a location that must be known to the general adapter. Enter this session location here. A session can support multiple connections.

Once you have entered the **session path** location, click on the pulldown arrow for the **connection name**, which will display a selection list of valid connections.

Session path* /bea/bse/sessions/default

Connection name* Securities

9. Click Connect to EIS. The Application View Administration page opens.

Note: You can access the Configure Connection Parameters page (displayed in the previous step) when the application view is not deployed, simply by selecting the Reconfigure connection parameters link. If the application view is deployed, you can access the page by first undeploying the application view.

Configuring Service Adapter Application Views

After you create and configure an application view, you can add services that support the application's functions. For information on creating the application view, see "Creating Service Adapter Application Views."

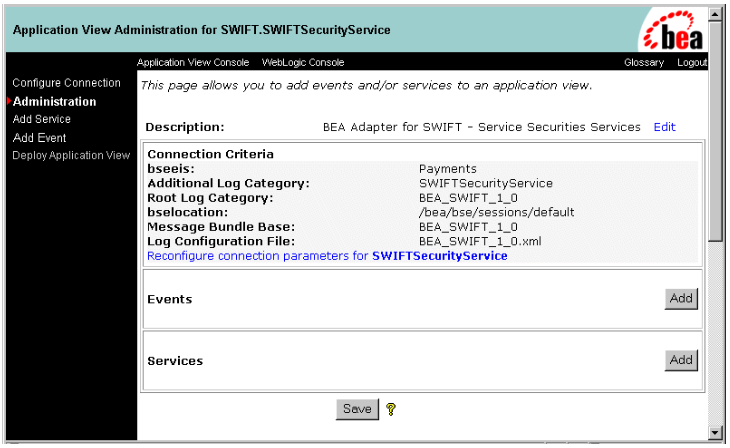
To add a service to an application view:

1. If it is not already open, open the application view to be modified.

2. For more information, see “Editing an Application View” in “Defining an Application View” in *Using Application Integration*:
   - For WebLogic Integration 7.0, see `http://edocs.bea.com/wli/docs70/aiuser/2usrdef.htm`
3. If the application view is deployed, you must undeploy it before adding the service. See “Optional Step: Undeploying an Application View” in “Defining an Application View” at the URL referenced in the previous step.
4. In the left pane, click Administration from the Configure Connection list.

   The Application View Administration window opens.

Figure 3-4 Application View Administration for SWIFT.SWIFTSecurityService



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

Description: BEA Adapter for SWIFT - Service Securities Services Edit

Connection Criteria

bseeis:	Payments
Additional Log Category:	SWIFTSecurityService
Root Log Category:	BEA_SWIFT_1_0
bselocation:	/bea/bse/sessions/default
Message Bundle Base:	BEA_SWIFT_1_0
Log Configuration File:	BEA_SWIFT_1_0.xml

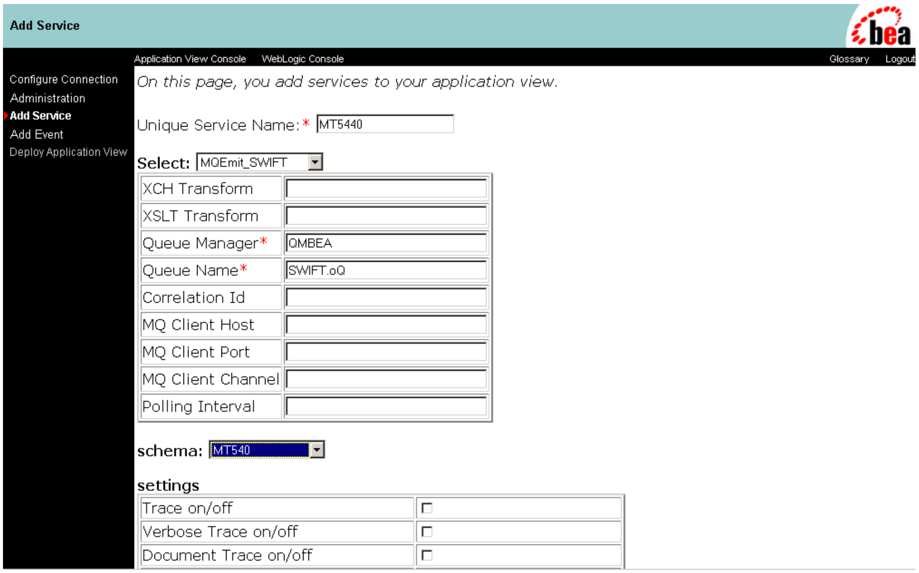
Reconfigure connection parameters for **SWIFTSecurityService**

1. From the Services pane of the Application View Administration page, select Add.

The Add Service Page opens. Choose from three protocols:

- MQEmit_SWIFT
- FileEmitter_SWIFT
- FTPEmitter_SWIFT

Figure 3-5 Add Service Window



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

Unique Service Name:* MT5440

Select: MQEmit_SWIFT

XCH Transform	
XSLT Transform	
Queue Manager*	QMBEA
Queue Name*	SWIFT.oQ
Correlation Id	
MQ Client Host	
MQ Client Port	
MQ Client Channel	
Polling Interval	

schema: MT540

settings

Trace on/off	☐
Verbose Trace on/off	☐
Document Trace on/off	☐

2. Enter the required properties of MQSeries as described in the following MQSeries Properties Table.

   The properties correspond to the MQSeries communication settings that the service adapter uses to communicate with MQSeries to write messages to the queues and with the transform options described in Chapter 4, "Transforming Document Formats."

Table 3-1 MQSeries Service Properties

Property	Description	Type	Sample Value	Element
Correlation Id	The correlation ID to set in the MQSeries message header.	duration		<correlid>
MQ Client Channel	For MQ Client only. Channel between an MQ Client and MQ Server.	string		<channel>

Table 3-1 MQSeries Service Properties (Continued)

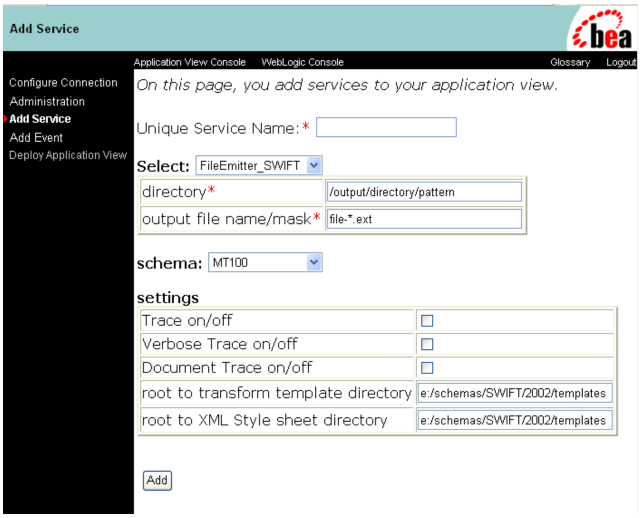
Property	Description	Type	Sample Value	Element
MQ Client Host	For MQ Client only. Host on which MQ Server is located.	string		<host>
MQ Client Port	For MQ Client only. Port number to connect to an MQ Server.	integer		<port>
Polling Interval	The time in milliseconds	duration		<timeout>
Queue Manager	Name of the MQSeries Queue Manager to be used.	string	OMBEA	<manager>
Request Queue Name	Queue on which request documents are received.	string	TEST.iO	<queue>
XCH Transform	Name of the `.xch` transform template file used by the XCH transform type in the XML to XML transformation phase. For more information, see Chapter 4, “Transforming Document Formats.”	string		<in_xmlg>

Table 3-1 MQSeries Service Properties (Continued)

Property	Description	Type	Sample Value	Element
XSLT Transform	Name of the XSLT stylesheet used by the XSLT transform type in the XML to XML transformation phase. **Note:** Do not enter an `.mfl` extension for a Message Format Language (MFL) file; these files are not stored with an extension. For more information, see Chapter 4, "Transforming Document Formats."	string		<in_xslt>

For the FileEmitter_SWIFT protocol, the following settings appear.

Figure 3-6 FileEmitter_SWIFT Properties



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

Unique Service Name:*

Select: FileEmitter_SWIFT

directory*	/output/directory/pattern
output file name/mask*	file-*.ext

schema: MT100

settings

Trace on/off	☐
Verbose Trace on/off	☐
Document Trace on/off	☐
root to transform template directory	e:/schemas/SWIFT/2002/templates
root to XML Style sheet directory	e:/schemas/SWIFT/2002/templates

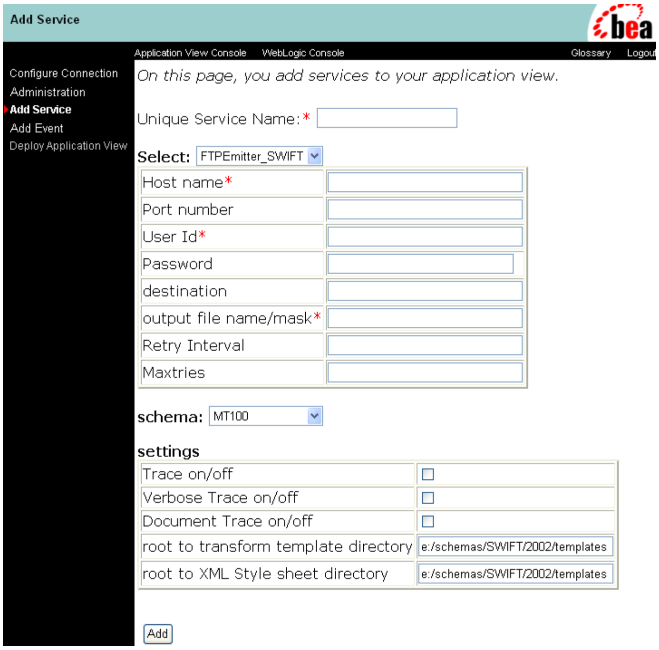
The following table presents the select properties at the beginning of the FileEmitter Swift interface.

Table 3-2 FileEmitter Select Properties

Property	Description
directory* (*Required)	**Type/Value:** Directory Path **Description:** Directory to which output messages are emitted.
output file name/mask* (*Required)	**Type/Value:** String **Description:** The output file name (can contain a '*'), which gets expanded to a timestamp. A pound symbol can be used as a mask for a sequence count. Each pound symbol represents a whole number integer value. For example, File## counts up to 99 before restarting at 0, File### counts up to 999 before restarting at 0, and so on.

For the FTPEmitter_SWIFT protocol, the following settings appear. The schema drop-down list corresponds to the manifest that describes all event schemas.

Figure 3-7 FTPEmitter_SWIFT Protocol



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

Unique Service Name:*

Select: FTPEmitter_SWIFT

Host name*	
Port number	
User Id*	
Password	
destination	
output file name/mask*	
Retry Interval	
Maxtries	

schema: MT100

settings

Trace on/off	☐
Verbose Trace on/off	☐
Document Trace on/off	☐
root to transform template directory	e:/schemas/SWIFT/2002/templates
root to XML Style sheet directory	e:/schemas/SWIFT/2002/templates

The following table describes the FTP Emitter SWIFT protocol settings.

Table 3-3 FTPEmitter_SWIFT Settings

Setting	Meaning/Properties
destination	**Type/Value:** String **Description:** Directory to address on the FTP target system.
Host name* (*Required)	**Type/Value:** String **Description:** FTP target system.
Maxtries	**Type/Value:** String **Description:** Number of retries for a failed attempt to write.
output file name/mask* (*Required)	**Type/Value:** String **Description:** The output file name (can contain a '*'), which gets expanded to a timestamp.

Table 3-3 FTPEmitter_SWIFT Settings

Setting	Meaning/Properties
Password* (*Required)	**Type/Value:** String **Description:** Password for the user account to use when connecting to the protocol host.
Port number	**Type/Value:** Numeric **Description:** FTP target system port (leave empty for FTP default).
Retry Interval	**Type/Value:** Retry interval duration in `xxH:xxM:xxS` format. (for example, 1H:2M:3S, which is 1 hour 2 minutes and 3 seconds) **Description:** The maximum wait interval between retries when a connection fails.
User Id* (*Required)	**Type/Value:** String **Description:** User account ID to use when connecting to the protocol host.

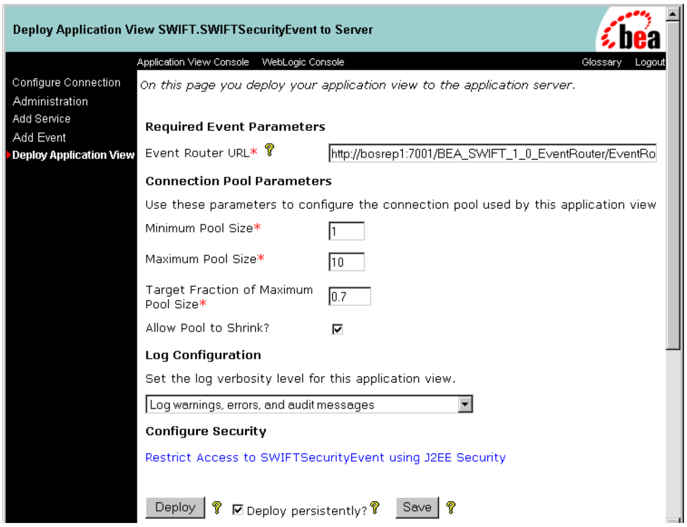
3. Choose from the following trace settings for service properties.

Table 3-4 Trace Settings for Services

Property	Description
Trace on/off	Generates a basic trace that displays the input XML (up to 300 bytes) before parsing, and shows the request being processed. For more information about tracing, see Chapter 7, *How Adapters Use Tracing*.
Verbose Trace on/off	Generates a trace that displays configuration parameters used by the adapter. For more information about tracing, see Chapter 7, *How Adapters Use Tracing*.
Document Trace on/off	Generates a trace that displays the input document after it was analyzed and the response document being returned. For more information about tracing, see Chapter 7, *How Adapters Use Tracing*.

4. Click Add and continue to the Deploy Application View page.

Figure 3-8 Deploy Application View


Deploy Application View SWIFT.SWIFTSecurityEvent to Server

On this page you deploy your application view to the application server.

Required Event Parameters

Event Router URL* http://bosrep1:7001/BEA_SWIFT_1_0_EventRouter/EventRo

Connection Pool Parameters

Use these parameters to configure the connection pool used by this application view

Minimum Pool Size* 1

Maximum Pool Size* 10

Target Fraction of Maximum Pool Size* 0.7

Allow Pool to Shrink? ✓

Log Configuration

Set the log verbosity level for this application view.

Log warnings, errors, and audit messages

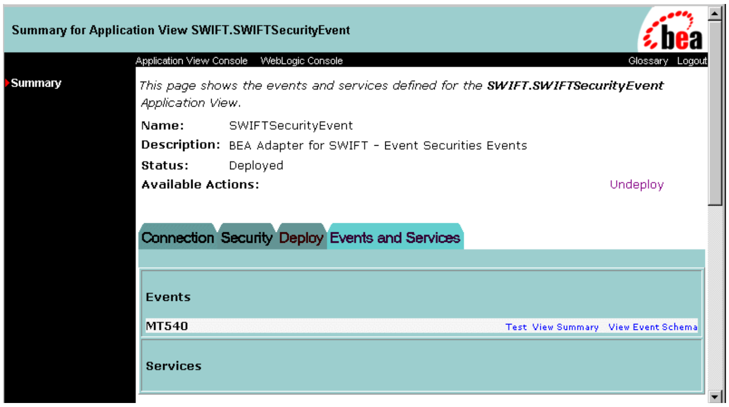
Configure Security

Restrict Access to SWIFTSecurityEvent using J2EE Security

5. Make any changes that you require to the Deploy Application View page and click Deploy.

   The Summary for Application View page opens on successful completion of the Application View Deploy.

Figure 3-9 Summary for Application View


Summary for Application View SWIFT.SWIFTSecurityEvent

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

Name: SWIFTSecurityEvent

Description: BEA Adapter for SWIFT - Event Securities Events

Status: Deployed

Available Actions: Undeploy

Events

MT540 Test View Summary View Event Schema

Services

Testing SWIFT Service Adapters

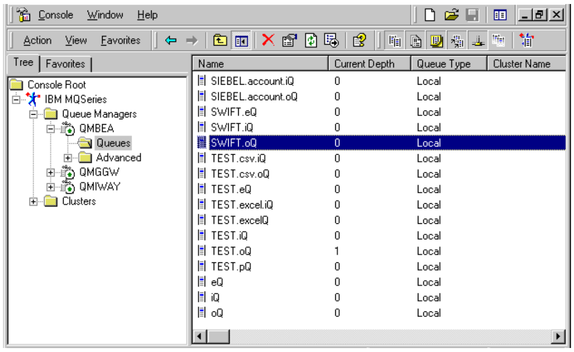
The service adapter emits a document to MQSeries and returns an emit status. You can validate the document’s arrival on the queue by browsing the MQSeries resources through IBM-supplied or custom tools. For example, on a Windows platform, you could use MQSeries Explorer, a Microsoft MMC plug-in utility. Using the MQSeries Explorer to connect to the Queue Manager and explore the queues on the Queue Manager, you can view the state of the queue and browse the messages in the queue.

1. Confirm that the queue is empty where the service adapter will send messages.

   For example, using MQSeries Explorer, browse the applicable queue and check the current queue depth, ensuring that it is zero.

   You can delete the existing queue messages by right-clicking and selecting All Tasks and then Clear Messages.

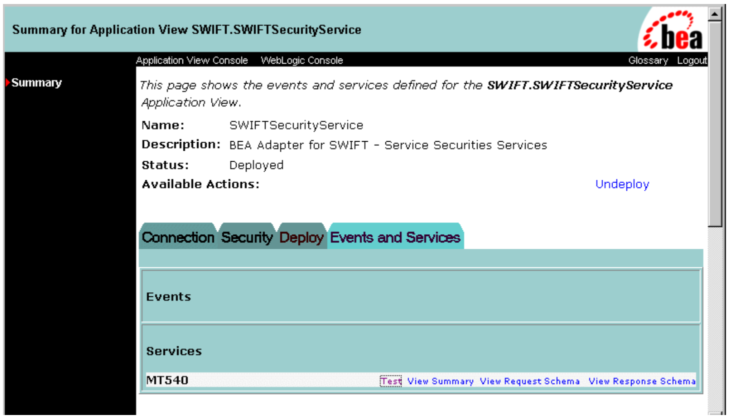
Figure 3-10 Confirming an Empty Output Queue



Name	Current Depth	Queue Type	Cluster Name
SIEBEL.account.iQ	0	Local	
SIEBEL.account.oQ	0	Local	
SWIFT.eQ	0	Local	
SWIFT.iQ	0	Local	
SWIFT.oQ	0	Local	
TEST.csv.iQ	0	Local	
TEST.csv.oQ	0	Local	
TEST.eQ	0	Local	
TEST.excel.iQ	0	Local	
TEST.excelQ	0	Local	
TEST.iQ	0	Local	
TEST.oQ	1	Local	
TEST.pQ	0	Local	
eQ	0	Local	
iQ	0	Local	
oQ	0	Local	

2. In the Summary for Application View page, click Test.

Figure 3-11 Summary for Application View - Test the Service


Summary for Application View SWIFT.SWIFTSecurityService

Application View Console WebLogic Console Glossary Logout

Summary

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

Name: SWIFTSecurityService
Description: BEA Adapter for SWIFT - Service Securities Services
Status: Deployed
Available Actions: Undeploy

Connection Security Deploy Events and Services

Events

Services

MT540 Test View Summary View Request Schema View Response Schema

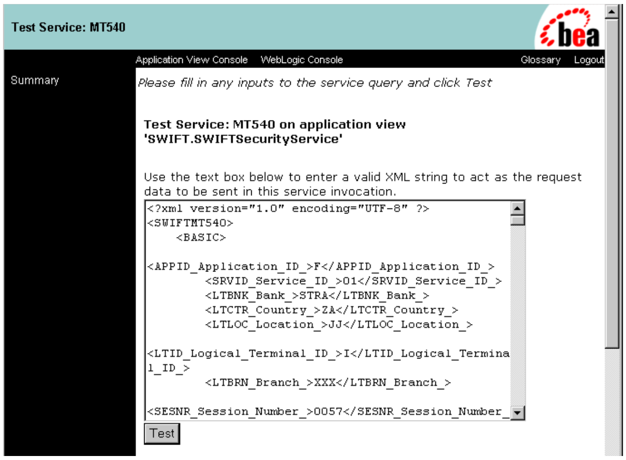
3. Enter a sample XML document that matches the request schema for the configured service. For example, the SWIFT MT540 request schema has an instance document similar to the following:

Figure 3-12 Instance Document

```
<?xml version="1.0" encoding="UTF-8"?>
<SWIFTMT540>
    <BASIC>
        <APPID_Application_ID_>F</APPID_Application_ID_>
        <SRVID_Service_ID_>01</SRVID_Service_ID_>
        <LTBNK_Bank_>STRA</LTBNK_Bank_>
      ...
            lines of document omitted
      ...
</SWIFTMT540>
```

4. Enter this document into the Service Test page by typing or by copying and pasting.

Figure 3-13 Test Service Window



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

Test Service: MT540 on application view 'SWIFT.SWIFTSecurityService'

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

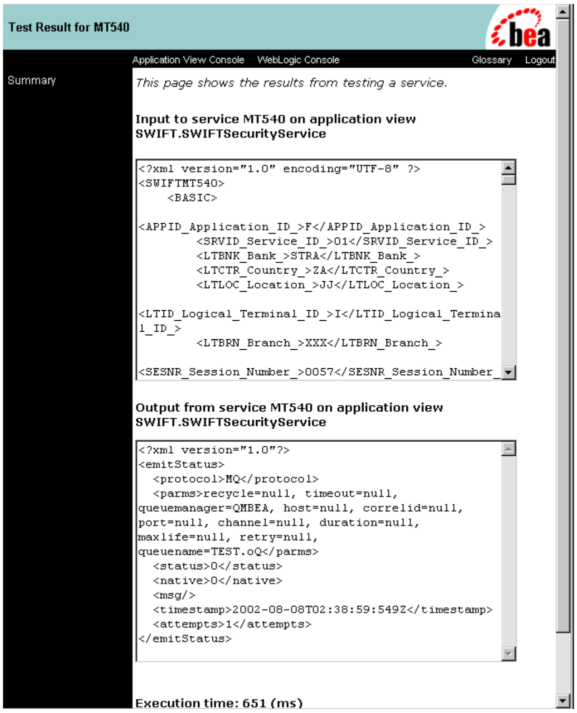
```
<?xml version="1.0" encoding="UTF-8" ?>
<SWIFTMT540>
    <BASIC>

<APPID_Application_ID_>F</APPID_Application_ID_>
        <SRVID_Service_ID_>01</SRVID_Service_ID_>
        <LTBNK_Bank_>STRA</LTBNK_Bank_>
        <LTCTR_Country_>ZA</LTCTR_Country_>
        <LTLOC_Location_>JJ</LTLOC_Location_>

<LTID_Logical_Terminal_ID_>I</LTID_Logical_Termina
l_ID_>
        <LTBRN_Branch_>XXX</LTBRN_Branch_>

<SESNR_Session_Number_>0057</SESNR_Session_Number_
```

5. Click Test to send the request through the service adapter to the IBM MQSeries Queue.

   The response document should indicate the success of emitting to MQSeries.

   See the following sample response.

Figure 3-14 Test Result Window - Request Successfully Sent


Test Result for MT540

Application View Console WebLogic Console Glossary Logout

Summary

This page shows the results from testing a service.

Input to service MT540 on application view SWIFT.SWIFTSecurityService

```
<?xml version="1.0" encoding="UTF-8" ?>
<SWIFTMT540>
    <BASIC>

<APPID_Application_ID_>F</APPID_Application_ID_>
        <SRVID_Service_ID_>01</SRVID_Service_ID_>
        <LTBNK_Bank_>STRA</LTBNK_Bank_>
        <LTCTR_Country_>ZA</LTCTR_Country_>
        <LTLOC_Location_>JJ</LTLOC_Location_>

<LTID_Logical_Terminal_ID_>I</LTID_Logical_Termina
l_ID_>
        <LTBRN_Branch_>XXX</LTBRN_Branch_>

<SESNR_Session_Number_>0057</SESNR_Session_Number_
```

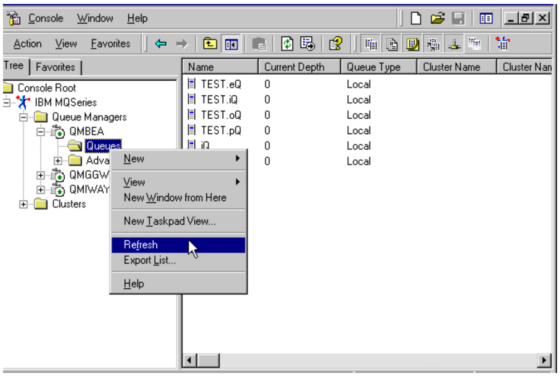
Output from service MT540 on application view SWIFT.SWIFTSecurityService

```
<?xml version="1.0"?>
<emitStatus>
  <protocol>MQ</protocol>
  <parms>recycle=null, timeout=null,
queuemanager=QMBEA, host=null, correlid=null,
port=null, channel=null, duration=null,
maxlife=null, retry=null,
queuename=TEST.oQ</parms>
  <status>0</status>
  <native>0</native>
  <msg/>
  <timestamp>2002-08-08T02:38:59:549Z</timestamp>
  <attempts>1</attempts>
</emitStatus>
```

Execution time: 651 (ms)

After transformation and formatting, the document should have arrived on the MQSeries queue.

Figure 3-15 MQSeries Queue



Name	Current Depth	Queue Type	Cluster Name	Cluster Nan
TEST.eQ	0	Local		
TEST.iQ	0	Local		
TEST.oQ	0	Local		
TEST.pQ	0	Local		
iQ	0	Local		
	0	Local		

6. Check for the output document on the queue designated in configuring the service.

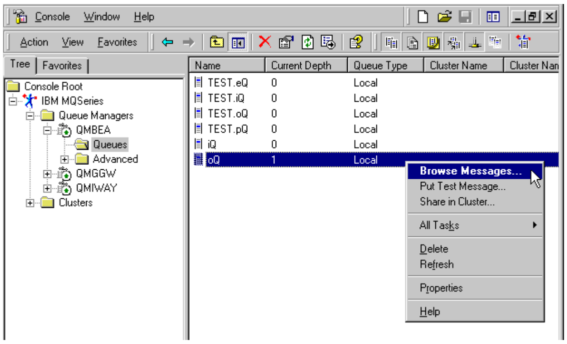
   a. Using SWIFT Explorer, browse to the appropriate queue.

   b. Select Refresh:

   The Current Depth should be incremented.

   There should be one if you cleared all messages first.

Figure 3-16 MQSeries Queue - Browse Messages



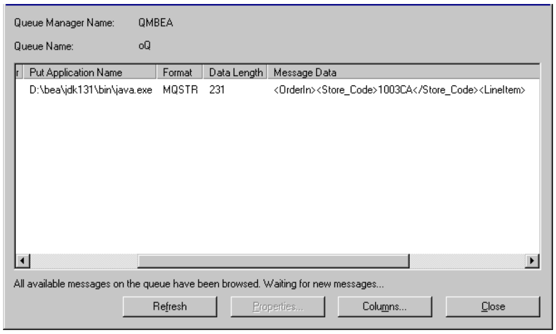
Name	Current Depth	Queue Type	Cluster Name	Cluster Nan
TEST.eQ	0	Local		
TEST.iQ	0	Local		
TEST.oQ	0	Local		
TEST.pQ	0	Local		
iQ	0	Local		
oQ	1	Local		

7. Select the appropriate queue.

8. Right-click and select Browse Messages.

   You should see a window similar to the following Message Browser with a column containing the data sent by the service adapter to SWIFT.

Figure 3-17 Message Browser Window



Queue Manager Name: QMBEA

Queue Name: oQ

r	Put Application Name	Format	Data Length	Message Data
	D:\bea\jdk131\bin\java.exe	MQSTR	231	<OrderIn><Store_Code>1003CA</Store_Code><LineItem>

All available messages on the queue have been browsed. Waiting for new messages...

4 Transforming Document Formats

This section describes how you can convert documents between XML and non-XML formats using several transformation phases. It includes the following topics:

- Transforming SWIFT to XML
- Transforming XML to XML
- Transforming XML to SWIFT

Documents within WebLogic Integration are encoded in XML. However, you may need to receive and generate non-XML data. BEA adapters support inbound transformations for the event adapter and outbound transformations for the service adapter. This chapter describes the transformation options available to you.

The BEA WebLogic Adapter for SWIFT provides SWIFT to XML transformation in the event adapter, XML to SWIFT in the service adapter, and generalized (xch or xslt) XML to XML transforms in both the event and service adapters.

WebLogic Integration supports several transformation phases for converting data from one format to another. Each phase offers several methods, or transforms, for accomplishing the conversion.

There are two transformation phases in an event adapter sending a message from an Enterprise Information System to business process management workflows.

1. SWIFT format to XML
2. XML to XML

There are two transformation phases in a service adapter sending a message from business process management workflows to a SWIFT message handling system.

1. XML to XML
2. XML to SWIFT format

You specify the type of transformation when adding an event or service to an application view.

Transforming SWIFT to XML

An event adapter that interfaces with SWIFT data sources must be able to convert that data to XML for processing using business process management workflows. This conversion includes “pre-parsing” the data into XML, which is then parsed itself for processing.

SWIFT format messages arriving at the event adapter are processed by the SWIFT pre-parser, which determines the SWIFT message type, and applies the correct SWIFT to XML format conversion.

Transforming XML to XML

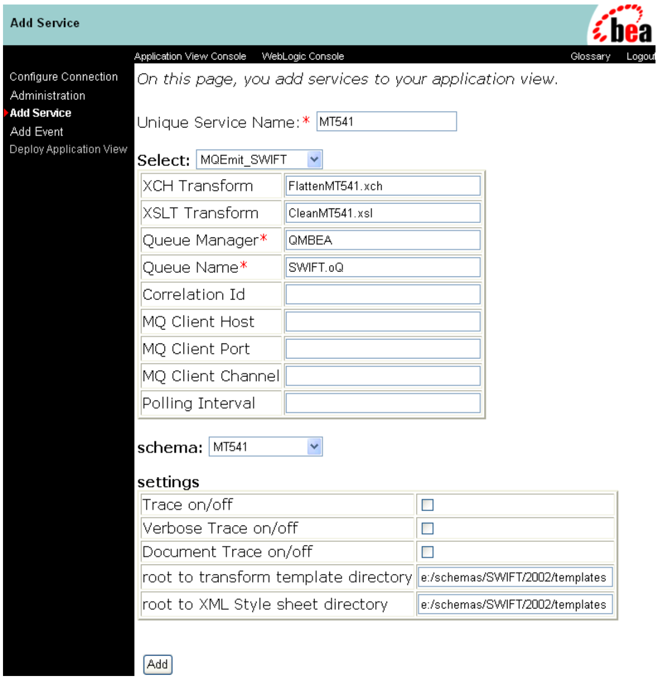
An event adapter or service adapter may be required to convert data from one type of XML document to another. You can choose between two types of transforms to accomplish this conversion:

- *XCH***,** a general conversion system which uses dictionaries (implemented as .dic files) and transformation templates (implemented as `.xch` files).
- *XSLT***,** a language for transforming XML documents into other XML documents. *XSLT* is part of XSL, the XML style sheet language.

You can invoke either or both transforms during this phase. If you specify both, XSLT occurs after XCH.

For example, the MT541 event specifies an XCH transform that uses the `FlattenMT541.xch` generalized transformation file and an XSLT transform that uses the `CleanMT541.xsl` style sheet.

Figure 4-1 Application View Console - Add Service



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

Unique Service Name:* MT541

Select: MQEmit_SWIFT

XCH Transform	FlattenMT541.xch
XSLT Transform	CleanMT541.xsl
Queue Manager*	QMBEA
Queue Name*	SWIFT.oQ
Correlation Id	
MQ Client Host	
MQ Client Port	
MQ Client Channel	
Polling Interval	

schema: MT541

settings

Trace on/off	☐
Verbose Trace on/off	☐
Document Trace on/off	☐
root to transform template directory	e:/schemas/SWIFT/2002/templates
root to XML Style sheet directory	e:/schemas/SWIFT/2002/templates

The parameters for each type of transform are as follows.

Table 4-1 XML to XML Transform Parameters

Transform Type	Parameter	Value/Description/Example
XCH Transform	transformation template filename	**Type/Value:** string **Description:** The name of the transformation template file including the `.xch` extension. **Example:** FlattenMT541.xch

Table 4-1 XML to XML Transform Parameters

Transform Type	Parameter	Value/Description/Example
XSLT Transform	XSLT style sheet filename	**Type/Value:** string **Description:** The name of the transformation template file including the `.xch` extension. **Example:** CleanMT541.xsl

Transforming XML to SWIFT

A service adapter that interfaces with SWIFT data sources must be able to convert XML documents (for example, from business process workflows) to SWIFT standard format. This conversion includes “pre-emitting” the data into the SWIFT, non-XML format, which is then emitted to the SWIFT message handling system (MHS).

The BEA WebLogic Adapter for SWIFT knows the document type by the root element of the XML representation of it. From the root node, the correct XML to SWIFT transformation is performed, and the non-XML SWIFT format is emitted on the configured service protocol (for example, to an MQSeries Queue or File directory).

5 Modifying Document Definitions

This section introduces schemas and schema repositories. It provides the steps for naming a schema repository, creating a repository manifest, and creating a schema. This section includes the following topics:

- Creating Schema Repositories
- Modifying the Repository
- Generating Transformation Templates and Document Schemas

SWIFT documents are described by dictionary files. They are converted into XML on entry to an event adapter and converted to SWIFT on exit from a service adapter. Transformation from SWIFT to XML is described in Chapter 4, "Transforming Document Formats" and is based on a generalized transformation engine making use of dictionaries and xch transformation templates stored in `.xch` files.

The XML representations of the SWIFT documents within WebLogic Integration system are described by corresponding W3C XML schemas. These schemas describe each event arriving to and propagating out of an event adapter. Schemas describe each request sent to and each response received from a service adapter. There is one schema for each event and two schemas for each service (one for the request and one for the response). The schemas are usually stored in files with an `.xsd` extension.

All BEA adapters make use of a schema repository to store their schema information and present it to the WebLogic Integration Application View Console. The schema repository is a directory or zip file that contains:

- A manifest file that describes the event and service schemas.
- The corresponding schema descriptions.

- A dictionary directory containing the SWIFT document dictionaries.
- A templates directory containing the transformation template for converting from and to SWIFT.
- A rules directory containing the content and network validation rules for the validation and acknowledgement process (see Chapter 1, "Acknowledgement Handling").
- A code set directory under rules to include the ISO code set for countries and currencies.

Creating Schema Repositories

The schema repository has a three-part naming convention.

`session_base_directory/adapter_type/connection_name`

here:

`session_base_directory`
: Is the schema's session base path that represents a folder where multiple sessions of schemas can be held.

`adapter_type`
: Is the type of adapter (for example, SWIFT or SAP).

`connection_name`
: Is a name representing a particular instance of the adapter type. For example, Securities may be a connection for a particular SWIFT system, and Payments may be another; each of these systems having different events and services relevant to them.

For example, if the session base path is `/usr/opt/bea/bse`, the adapter type is SWIFT, and the connection name is category5, then the schema repository is the directory:

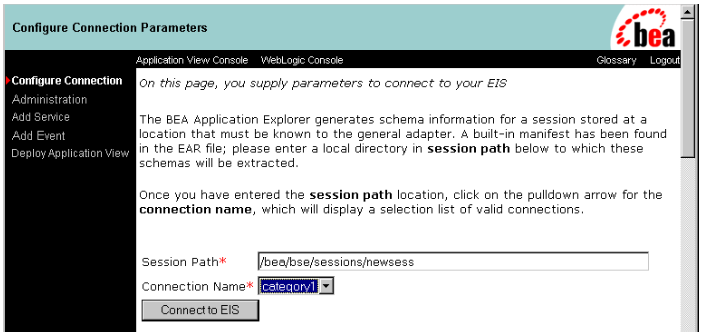
```
/usr/opt/bea/bse/SWIFT/category5
```

The BEA WebLogic Adapter for SWIFT comes pre-delivered with a repository for all SWIFT Category 5 documents. During creation of an application view, this repository is extracted from the supplied `BEA_SWIFT_1_0.ear` file into the directory you choose. Each time a new application view is created, the repository is extracted from the EAR file.

Modifying the Repository

During creation of the application view, the Session Path is supplied (see Figure 5-1), and any supplied connection repositories are extracted to the location you specify.

Figure 5-1 Configure Connection Parameters



Configure Connection Parameters

On this page, you supply parameters to connect to your EIS

The BEA Application Explorer generates schema information for a session stored at a location that must be known to the general adapter. A built-in manifest has been found in the EAR file; please enter a local directory in **session path** below to which these schemas will be extracted.

Once you have entered the **session path** location, click on the pulldown arrow for the **connection name**, which will display a selection list of valid connections.

Session Path* /bea/bse/sessions/newsess

Connection Name* category1

If a built-in manifest is found in the EAR file, it is extracted to the location you specify. Any duplicate files are overwritten. The Connection name drop-down list box reflects the contents of the directory after extract; that is, the contents of the built-in repository described by the built-in manifest and any existing repository connections.

There are two methods for making changes to the repository available to an application view:

- Modify repository on disk.
- Modify repository and update EAR file.

Modify Repository on Disk

Modifications made on disk in the session path supplied to the application view are preserved if they are in a directory representing a session that is not supplied with the EAR file. This first procedure allows changes made to the document descriptions, schemas, transformation templates, rules, and rule code sets by ensuring that they exist in a directory that will not be overwritten.

To make changes to the repository:

1. Copy or create a directory under the “session_path/SWIFT” location to reflect your desired new custom connection repository (for example, Category5_Custom).
2. Create a new application view supplying the same session path. At this point, the new connection (for example, Category5_Custom) appears in the list and will not be overwritten by the original supplied connection repository.

Modify Repository and Update EAR File

This procedure ensures that modifications become a permanent feature of the product by updating the supplied EAR file.

To make changes to the repository:

1. Modify objects in the repository.
2. Load the EAR file with the changed repository.
   a. From the supplied EAR file, extract the `BEA_SWIFT_1_0.jar` file.
   b. From the extracted `BEA_SWIFT_1_0.jar` file, extract the `BEA_SWIFT_1_0.manifest.zip`.
   c. Replace/Update the files in `BEA_SWIFT_1_0.manifest.zip` with the modified objects (`manifest.xml`, dictionaries, transformation templates, rules, and code sets).
   d. Replace/Update the `BEA_SWIFT_1_0.manifest.zip` in the `BEA_SWIFT_1_0.jar`.
   e. Replace the `BEA_SWIFT_1_0.jar` in the `BEA_SWIFT_1_0.ear`.

At this point, whenever a new application view is created, your modified connection repository will be extracted out of the `BEA_SWIFT_1_0.ear` (from the included `BEA_SWIFT_1_0.jar`, from the `BEA_SWIFT_1_0.manifest.zip`).

Generating Transformation Templates and Document Schemas

A sample utility class is provided with the BEA WebLogic Adapter for SWIFT to automatically generate a transformation template from a SWIFT document described in a dictionary (`.dic` file). A second utility class is provided to generate a W3C XML schema for the related XML document. Finally, a command script is provided to automatically generate all Transformation Templates and all W3C XML schemas for all dictionaries in a session repository. This section describes the procedure for making use of these sample utilities.

Prerequisites for the Sample Utilities

The two utility classes that create the transformation templates and create the document schema rely on the Sun Java XML Support Pack. This package is included in the J2SDK version 1.4, or as an add-on in J2SDK v1.2 and v1.3. If you are not using v1.4 of the Java SDK, you must obtain the XML Pack from Sun Microsystems at `http://java.sun.com/xml/javaxmlpack.html`. The JAXP component of this pack must be available to the java runtime and therefore, must be added to the class path (not necessary for J2SDK v1.4).

Extracting the Sample Utilities

The sample utilities are packed with the associated SAMPLES zip file for the BEA WebLogic Adapter for SWIFT. To make use of these utilities, you must extract the samples directory from the zip file. From the directory containing the zip file, use WinZip or the following java jar command to extract the samples subdirectory:

```
jar -xvf BEA_SWIFT_1_0.SAMPLES.zip samples
```

This command creates a samples subdirectory at the current location.

Generating Transformation Templates

The supplied utility class `XCHZen.class` generates transformation templates to transform from SWIFT to XML and from XML to SWIFT. The utility class is called by the following commands:

- To generate a SWIFT to XML template:

  ```
  java XCHZen -os NT -NONXMLtoXML -dic YourDictionary.dic -xch
  ../templates/YourDictionarytoXML.xch
  ```

- To generate an XML to SWIFT template:

  ```
  java XCHZen -os NT -XMLtoNONXML -dic YourDictionary.dic -xch
  ../templates/XMLtoYourDictionary.xch
  ```

here *`YourDictionary`* is the name of the dictionary name (for example, MT540).

Generating Document Schemas

The supplied utility class `XSDZen.class` generates W3C XML schemas for the XML representations of the SWIFT documents. These schemas are based on the standard transformation performed by the generated transformation templates in the previous section, not on any custom mapping that a user may have performed. The utility class is called by the following command:

```
java XSDZen -noheader -dic dictionaries/YourDictionary -xsd
YourDictionary.xsd
```

here *`YourDictionary`* is the name of the dictionary name (for example, MT540).

Automatically Generating a Session Repository

The utility genswift.cmd supplied in the samples directory automatically regenerates all transformation templates and document schemas for all sessions in a session path (see“Extracting the Sample Utilities” on page 5-5 for instructions on extracting samples from the EAR file). In other words, from the session path, all directories under `session_path/SWIFT` are traversed and all dictionaries in the dictionary directory generate an XML to SWIFT template, a SWIFT to XML template, and a document schema for the XML representation.

From the session path (see Figure 5-1) enter the following command:

```
samples_path/genswift.cmd
```

here *`samples_path`* is the location of the samples directory extracted in the section “Extracting the Sample Utilities” on page 5-5. At this point, you may follow the instructions in “Modify Repository and Update EAR File” on page 5-4 to make the results permanent for your product.

CHAPTER

1 Acknowledgement Handling

Documents received by the BEA WebLogic Adapter for SWIFT are processed in stages that include preparse, validate, transform, agent execute, and pre-emit. At any phase, the document may generate errors and may or may not pass specific validation rules.

The validation engine and document validation rules are described in full in Appendix A, "Rules System Adapter." This section describes the process of acknowledging a document after it has passed through validation and includes the following topics:

- Acknowledgement Processing
- Documents, Validation, and Acknowledgement
- Acknowledgement Agent
- Acknowledgement Message Handling

Acknowledgement Processing

The acknowledgement process responds to receipt of a document to indicate the receipt and validity of the received document. The features of the adapter which support the acknowledgement process are:

- Validation

Validation is a specific stage in processing the document that occurs immediately after the document arrives and is available in XML format (that is, after the XML structure is available but before any other processing). The process of validation and the rules used in validating a document are described in Appendix A, “Rules System Adapter.”

- Document Tree

  The document tree is the adapter’s representation of the XML document. The tree is used during document processing and stores additional document or document / element level information. Validation errors are stored in the document tree and are available to the acknowledgement agent.

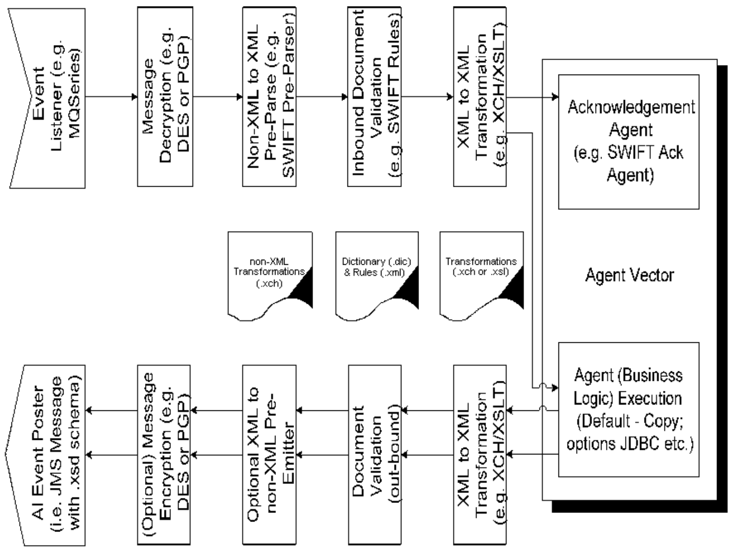
- Acknowledgement Agent

  The acknowledgement agent is responsible for determining what processing should occur for a document, as represented in memory by the document tree, and contains zero or one or more validation errors. The agent determines what constitutes a good document, to which an “ACK” or acknowledgement should be sent. The agent also determines what constitutes a bad document, to which a “NAK” or non-acknowledgement should be sent. The agent also determines the content of the ACK and the NAK.

Documents, Validation, and Acknowledgement

Documents proceed from the Adapter Event Listener to the Emitter or Event Poster and are processed in stages. This processing is show in the following diagram:

Figure 1-1 Document Life Cycle


Event Listener (e.g. MQSeries)
Message Decryption (e.g. DES or PGP)
Non-XML to XML Pre-Parse (e.g. SWIFT Pre-Parser)
Inbound Document Validation (e.g. SWIFT Rules)
XML to XML Transformation (e.g. XCH/XSLT)
Acknowledgement Agent (e.g. SWIFT Ack Agent)
non-XML Transformations (.xch)
Dictionary (.dic) & Rules (.xml)
Transformations (.xch or .xsl)
Agent Vector
Agent (Business Logic) Execution (Default - Copy; options JDBC etc.)
AI Event Poster (i.e. JMS Message with .xsd schema)
(Optional) Message Encryption (e.g. DES or PGP)
Optional XML to non-XML Pre-Emitter
Document Validation (out-bound)
XML to XML Transformation (e.g. XCH/XSLT)

With respect to validation and acknowledgement, the above document life cycle has the following characteristics:

- Validation occurs as soon as the document has been converted into XML.
- Validation comprises both structural validation (described in dictionaries) and content & network validation (described in `Rules.xml` files).
- The validation processor (class) is defined at the document level or at the rule level. (See Appendix A, "Rules System Adapter" for a description.) An example validation processor is the XDSWIFTRules.class.
- Validation (particular Dictionary and Rules.xml) is tied to a specific XML document type (as defined in the Deploy.xml).
- If validation is defined for an XML document, an acknowledgement agent is added to the Agent Vector (first in line) for processing during the document's Agent Execution phase. See "Acknowledgement Agent" on page 1-4 for a discussion of acknowledgement agent determination.
- Validation processing adds error elements into the document tree.
- The acknowledgement agent processes the document via its document tree including any validation errors added during the previous validation phase.

- The output of the acknowledgement agent is independent of the output of the document agent (that is, different schema, separate thread of execution, and so forth).
- Validation errors and document output are independent of one another. In other words, a document may fail validation rules and have acknowledgements (ACK or NAK) generated in the acknowledgement agent. However, the document is still passed to its agent and sent to its output unless specific actions are taken (that is, logic is coded). For example, if the copy agent is defined for the particular document life cycle, the document is passed out and posted to the event router even if there are errors.
- Custom agents may be coded to alter behavior when validation errors are present in the document tree.

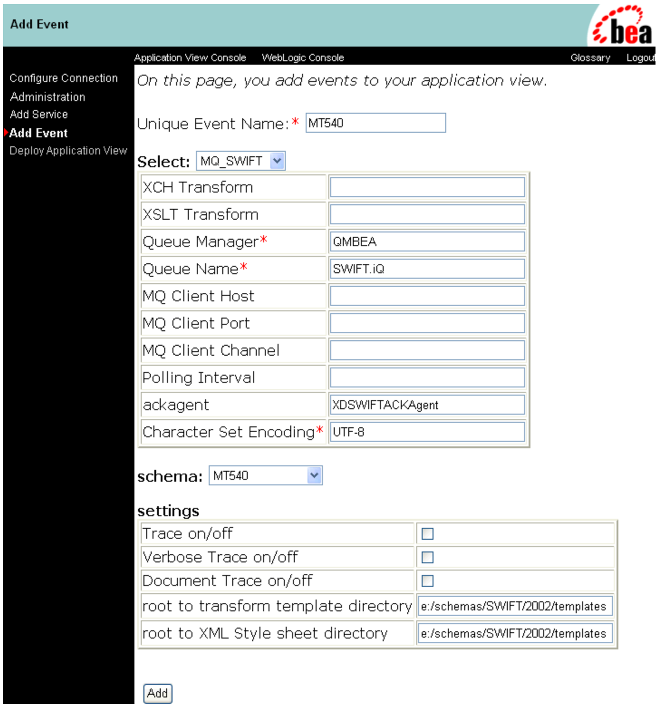
Acknowledgement Agent

The acknowledgement agent is defined by an <ackagent> entry and can be defined at the event or service level, in the document section of the `Deploy.xml` or as an attribute in the root element of the `Rules.xml` file. The search order is as follows:

1. The <document> section of the `deploy.xml`.
2. An attribute of the root element of the `Rules.xml` file.
3. The event or service listener definition via the Application View Console.

For the BEA WebLogic Adapter for SWIFT, the default ackagent is the XDSWIFTACKAgent. This is set or changed on the event or service configuration screen.

Figure 1-2 Edit Event Window



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

Unique Event Name:* MT540

Select: MQ_SWIFT

XCH Transform	
XSLT Transform	
Queue Manager*	QMBEA
Queue Name*	SWIFT.iQ
MQ Client Host	
MQ Client Port	
MQ Client Channel	
Polling Interval	
ackagent	XDSWIFTACKAgent
Character Set Encoding*	UTF-8

schema: MT540

settings

Trace on/off	☐
Verbose Trace on/off	☐
Document Trace on/off	☐
root to transform template directory	e:/schemas/SWIFT/2002/templates
root to XML Style sheet directory	e:/schemas/SWIFT/2002/templates

Acknowledgement Message Handling

The validation engine performs the structural, content, and network validation rules defined in the document specific dictionaries and `rules.xml` files. The acknowledgement agent generates an acknowledgement message based on the document tree which is a composite of the original XML document tree and any validation errors added by the validation engine. The results of the acknowledgement agent are dependent on the logic coded in the implementation class. For the BEA WebLogic Adapter for SWIFT, the default implementation class is the `XDSWIFTACKAgent.class`.

The following is a sample output with errors.

Listing 1-1 Sample Output

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

<eda>

       <error code="-103" source="validator"
timestamp="2002-08-08T17:37:34Z">

              Document failed validation: XD[FAIL] validation
error: checkList
[SWIFTMT541._541.E.E3.L_19A._19A.CURCD_Currency_Code_]: code is
missing

       </error>

</eda>
```

The following is a schema for the results of this acknowledgement, the XDSWIFTACKAgent.xsd.

Listing 1-2 XDSWIFTACKAgent.xsd

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">

       <xs:element name="Error" type="xs:string"/>

       <xs:element name="SWIFTack">

              <xs:complexType>

                     <xs:sequence>

                            <xs:element ref="Error" minOccurs="0"
maxOccurs="unbounded"/>

                     </xs:sequence>

              </xs:complexType>

       </xs:element>

</xs:schema>
```

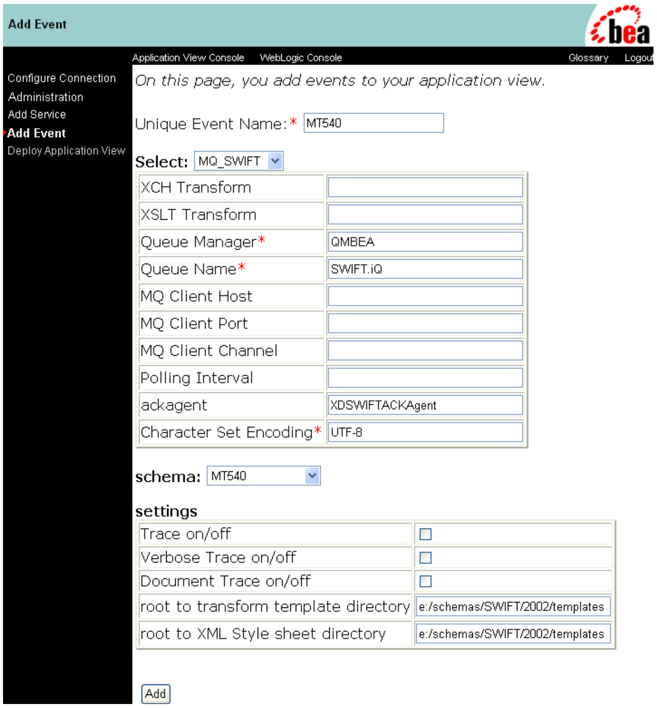
The acknowledgement message is generated separately from the document message. After the acknowledgement agent completes execution, there are two messages traversing the system that attempt to be posted to the event router. For the message to be posted, an event must be registered in the application view with the acknowledgement schema.

Creating an Acknowledgement Event

In addition to the application view event created for the document, there must be an event created for the acknowledgement message generated by the acknowledgement agent. The following procedure creates an event for the acknowledgements generated by the XDSWIFTACKAgent.

1. Add an event in the WebLogic Application View of the event adapter.

Figure 1-3 Add Event Window - Adding an ACK_NAK Event



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

Unique Event Name:* MT540

Select: MQ_SWIFT

XCH Transform	
XSLT Transform	
Queue Manager*	QMBEA
Queue Name*	SWIFT.iQ
MQ Client Host	
MQ Client Port	
MQ Client Channel	
Polling Interval	
ackagent	XDSWIFTACKAgent
Character Set Encoding*	UTF-8

schema: MT540

settings

Trace on/off	☐
Verbose Trace on/off	☐
Document Trace on/off	☐
root to transform template directory	e:/schemas/SWIFT/2002/templates
root to XML Style sheet directory	e:/schemas/SWIFT/2002/templates

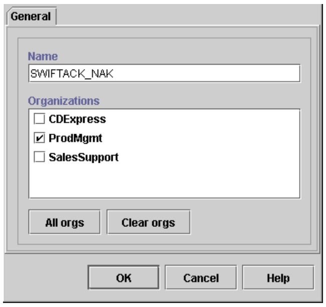
Note: The ackagent value is set to the desired acknowledgement agent (in this example, the XDSWIFTACKAgent). Additionally, there is an ACKNAK schema in the Schema drop down list box which handles both clean ACKs and error-filled NAKs.

It is important that the event adapter's protocol settings (in this case, MQSeries based) are identical to those provided for the original event. If the settings are different, a separate Event Listener is created, and the two events (document and associated ACK/NAK) are not tied together. The ACKs/NAKs created by the document will not be seen by the event or schema combination created in this section.

2. Add, continue, and deploy the application view.

3. From WebLogic Integration Studio, create a new workflow template.

Figure 1-4 Template Properties Dialog Box - Creating an ACK/NAK Template

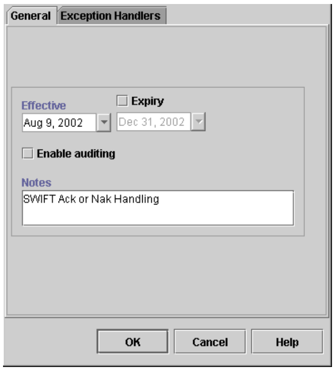


4. Type a name for the template to indicate this workflow is for the acknowledgement message.

5. Click OK.

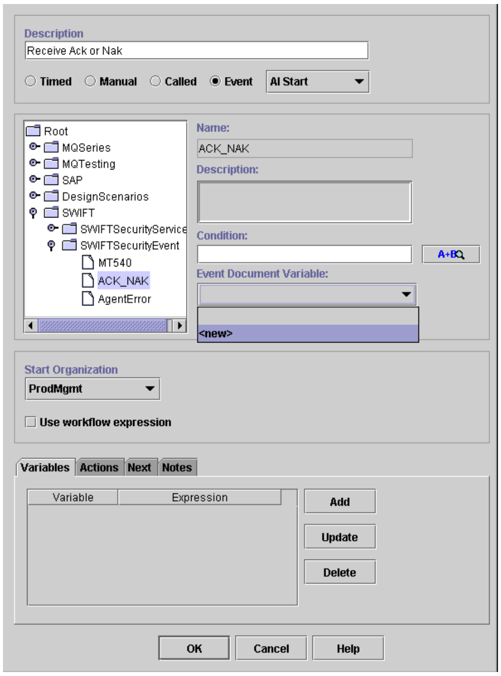
6. For the new template, create a new template definition.

Figure 1-5 Creating an ACK/NAK Template Definition



7. Open the new template definition, select the Start object, and enter the properties.

Figure 1-6 Start Properties Dialog Box - Event Definition



Description: Receive Ack or Nak

Timed / Manual / Called / [✓ Event] — AI Start

Root
- MQSeries
- MQTesting
- SAP
- DesignScenarios
- SWIFT
  - SWIFTSecurityService
  - SWIFTSecurityEvent
    - MT540
    - ACK_NAK
    - AgentError

Name: ACK_NAK

Description:

Condition:

Event Document Variable: <new>

Start Organization: ProdMgmt

Use workflow expression

a. Select Event →AI Start.

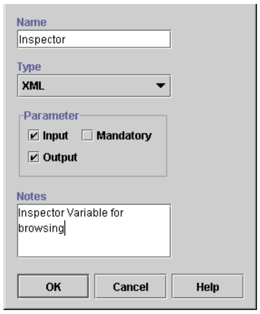
b. Select the SWIFT→SWIFTSecurityEvent→ACK_NAK event in the left hand AI event pane.

c. Choose the Start Organization to be the same as the template definition organization.

d. Add a new Event Document variable.

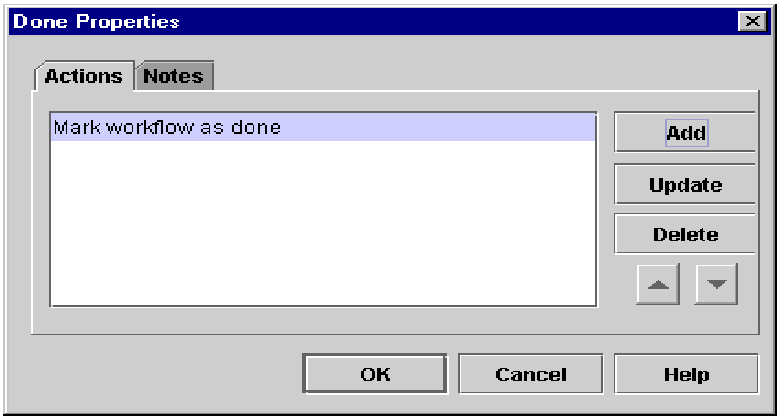
The Variables Properties dialog box appears.

Figure 1-7 Variable Properties Dialog Box - Add Start Variable


Name: Inspector
Type: XML
Parameter: [✓ Input] Mandatory [✓ Output]
Notes: Inspector Variable for browsing

8. Select Input and Output parameter type.
9. Click OK.
10. Add an Action to the Done object.
    a. Select the Done object.
    b. Click Add an Action.

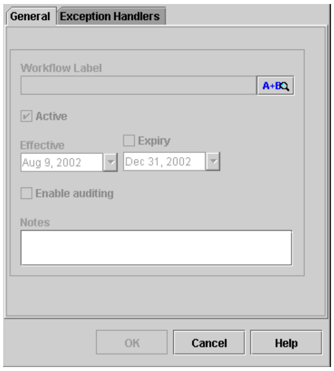
Figure 1-8 Done Properties Dialog Box - Marking Workflow as Done


Done Properties

Mark workflow as done

11. Click the Actions tab and select Mark Workflow as Done.
12. Click OK.
13. Right-click Message Definition in the left pane and select Save.

Figure 1-9 Template Definition Properties



Workflow Label:

[✓ Active]

Effective: Aug 9, 2002

[ ] Expiry: Dec 31, 2002

[ ] Enable auditing

Notes:

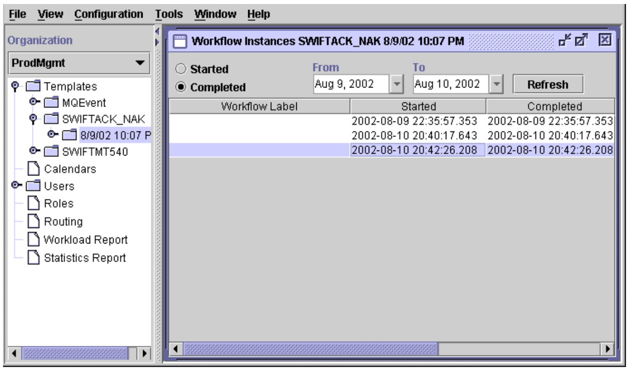
14. Ensure the workflow is active by selecting the properties of the workflow definition.

Testing Acknowledgement Message Handling

Having created a SWIFT event adapter with two registered events, a SWIFT message event (for example, SWIFT MT540) and a SWIFT ACK/NAK message event, you can view the document as it was processed through the workflow in the WebLogic Studio console.

1. Right-click Template and select Instances.

Figure 1-10 WebLogic Integration Studio - Template Workflow Instances



Workflow Instances SWIFTACK_NAK 8/9/02 10:07 PM

Started [✓ Completed]

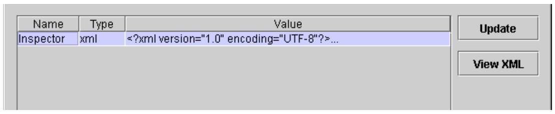
From: Aug 9, 2002 To: Aug 10, 2002

Workflow Label	Started	Completed
	2002-08-09 22:35:57.353	2002-08-09 22:35:57.353
	2002-08-10 20:40:17.643	2002-08-10 20:40:17.643
	2002-08-10 20:42:26.208	2002-08-10 20:42:26.208

2. Select the instance of interest (that is, the instance generated by the bad message).

3. Right-click the instance and select Workflow Variables:

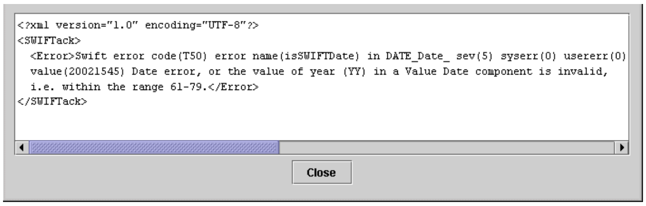
Figure 1-11 Workflow Variables



Name	Type	Value
Inspector	xml	<?xml version="1.0" encoding="UTF-8"?>...

4. Click View XML to see the contents of the XML variable “Inspector”:

Figure 1-12 Workflow XML Variable



```
<?xml version="1.0" encoding="UTF-8"?>
<SWIFTack>
  <Error>Swift error code(T50) error name(isSWIFTDate) in DATE_Date_ sev(5) syserr(0) usererr(0)
  value(20021545) Date error, or the value of year (YY) in a Value Date component is invalid,
  i.e. within the range 61-79.</Error>
</SWIFTack>
```

1 Using Tracing

Tracing is an essential feature of an adapter. Most adapters 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 a warning condition, the adapter cannot stop processing and wait for an end user to respond.

Moreover, many business applications that are connected by adapters are mission-critical. For example, an adapter might maintain an audit report of every transaction with an EIS. Consequently, adapter components must provide both accurate logging and auditing information. The adapter tracing and logging framework is designed to accommodate both logging and auditing.

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

- Levels and Categories of Tracing
- Tracing and Performance
- Creating Traces for Services and Events

Levels and Categories of Tracing

Tracing is provided by both the BEA adapter framework and by the BEA WebLogic Adapter for SWIFT. The BEA WebLogic Integration framework provides five distinct levels of tracing:

Table 1-1

Level	Indicates
AUDIT	An extremely important log message related to the business processing performed by an adapter. Messages with this priority are always written to the log.
ERROR	An error in the adapter. Error messages are internationalized and localized for the user.
WARN	A situation that is not an error, but that could cause problems in the adapter. Warning messages are internationalized and localized for the user.
INFO	An informational message that is internationalized and localized for the user.
DEBUG	A debug message, that is, information used to determine how the internal components work. Debug messages usually are not internationalized.

The adapter framework provides three specialized categories of tracing:

Table 1-2

Level	Indicates
Basic Trace	Basic traces. Displays the input XML (up to 300 bytes) before parsing, and shows the request being processed. The default setting is off.
Verbose Trace	More extensive traces. Displays configuration parameters used by the adapter. The default setting is off.

Table 1-2

Level	Indicates
Document Trace	Displays the input document after it was analyzed and the response document being returned. Because some documents are very large, this trace category can severely affect performance and memory use. The default setting is off.

Note: To obtain the appropriate trace, both the level and the category must be declared. In a debug situation, BEA Customer Support will request (minimally) a Basic and a Verbose trace.

Tracing and Performance

The additional trace capabilities provided by the adapter are not strictly hierarchic; rather they are categorized. These traces are designed to provide debugging help with minimum effect on performance. All internal adapter traces are controlled through the additional tracing 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 trace settings, the traces are generated but never appear in output. This affects performance, as the production of the trace continues even though you receive no benefit of the additional trace information.

Creating Traces for Services and Events

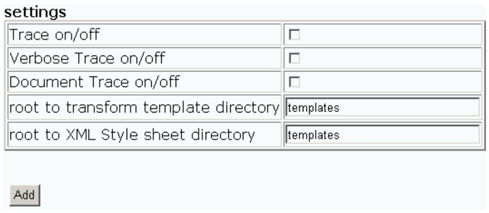
This following topics discuss the steps required to create traces to diagnose adapter problems.

Creating Traces for a Service

To create traces for a service:

1. Create or modify the service.
2. Ensure that all of the adapter parameters are entered correctly.

Figure 1-1 Add Service Window



settings

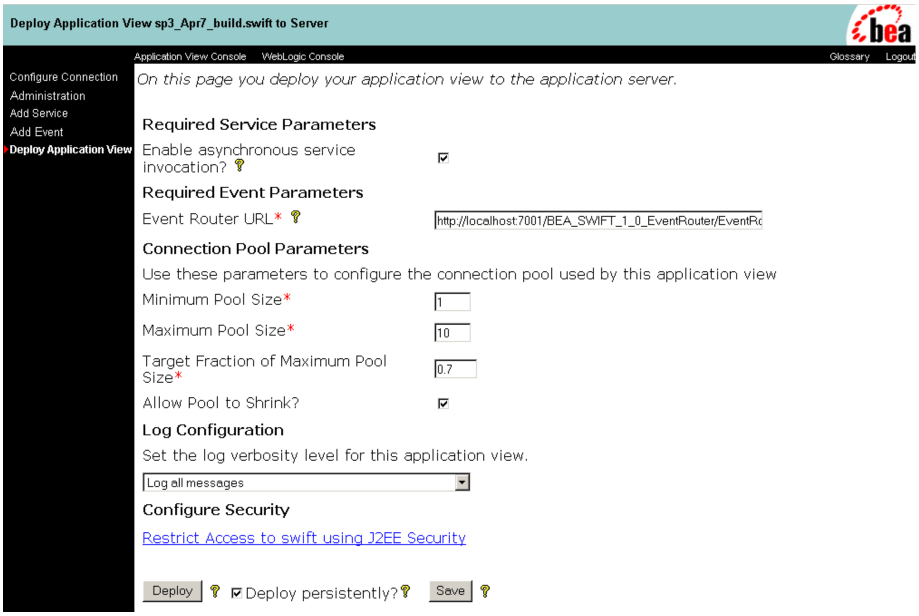
Trace on/off	☐
Verbose Trace on/off	☐
Document Trace on/off	☐
root to transform template directory	templates
root to XML Style sheet directory	templates

3. Select the appropriate schema from the drop-down list.
4. Select the appropriate trace levels as described in Table 1-2: Trace, Verbose trace, and Document trace.
5. Click Add to continue to the next configuration pane.
6. Click Continue to move to the next configuration pane.

   The Deploy Application View window opens.
7. Navigate to the Log Configuration area and select the desired trace level.

   This pane enables you to select the trace level for the BEA WebLogic Integration framework.

Figure 1-2 Deploy Application View window



Deploy Application View sp3_Apr7_build.swift to Server

On this page you deploy your application view to the application server.

Required Service Parameters

Enable asynchronous service invocation? ☑

Required Event Parameters

Event Router URL* http://localhost:7001/BEA_SWIFT_1_0_EventRouter/EventRo

Connection Pool Parameters

Use these parameters to configure the connection pool used by this application view

Minimum Pool Size* 1

Maximum Pool Size* 10

Target Fraction of Maximum Pool Size* 0.7

Allow Pool to Shrink? ☑

Log Configuration

Set the log verbosity level for this application view.

Log all messages

Configure Security

Restrict Access to swift using J2EE Security

☑ Deploy persistently?

For maximum tracing, select Log all Messages.

This is recommended to obtain optimum debugging information for BEA support personnel.

> **Note:** This causes all generated messages to be written to the log. You must select the desired category as defined in Table 1-2 in the adapter to generate the required messages.

8. Click Deploy (or Save) to set the trace settings and deploy the application view.

   Traces are created the next time the service is invoked.

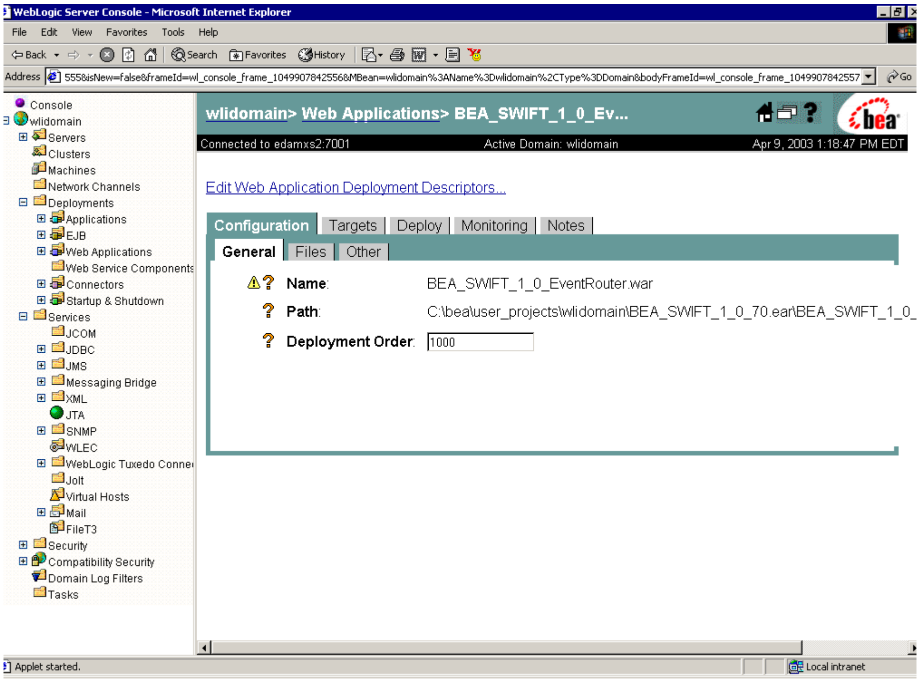
   Traces are output to a file named BEA_SWIFT_1_0.log in the WebLogic Domain home directory.

Creating or Modifying Framework Tracing Levels

To create or modify the WebLogic framework tracing level for an event:

1. Logon to the BEA WebLogic Server Console.

Figure 1-3 WebLogic Server Console


WebLogic Server Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address 555&isNew=false&frameId=wl_console_frame_1049907842556&MBean=wlidomain%3AName%3Dwlidomain%2CType%3DDomain&bodyFrameId=wl_console_frame_1049907842557
Go
Console
wlidomain
Servers
Clusters
Machines
Network Channels
Deployments
Applications
EJB
Web Applications
Web Service Components
Connectors
Startup & Shutdown
Services
JCOM
JDBC
JMS
Messaging Bridge
XML
JTA
SNMP
WLEC
WebLogic Tuxedo Conne
Jolt
Virtual Hosts
Mail
FileT3
Security
Compatibility Security
Domain Log Filters
Tasks
wlidomain> Web Applications> BEA_SWIFT_1_0_Ev...
bea
Connected to edamxs2:7001
Active Domain: wlidomain
Apr 9, 2003 1:18:47 PM EDT
Edit Web Application Deployment Descriptors...
Configuration
Targets
Deploy
Monitoring
Notes
General
Files
Other
Applet started.
Local intranet

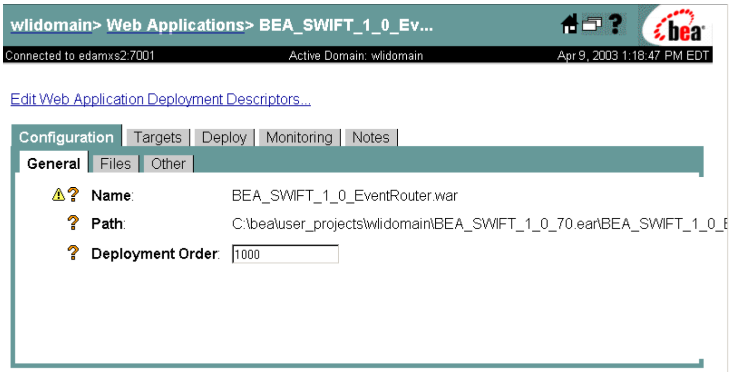
Name: BEA_SWIFT_1_0_EventRouter.war

Path: C:\bea\user_projects\wlidomain\BEA_SWIFT_1_0_70.ear\BEA_SWIFT_1_0_

Deployment Order: 1000

2. Select Web Applications.
3. Select BEA_SWIFT_1_0_EventRouter.war.
4. Click Edit Web Application Deployment Descriptors.
5. When the following window opens, select Servlets.
6. In the folder below Servlets, select EventRouterServlet.
7. Select Parameters.
8. Select LogLevel.

Figure 1-4 WebLogic Server Console: Configuration


wlidomain> Web Applications> BEA_SWIFT_1_0_Ev...
Connected to edamxs2:7001
Active Domain: wlidomain
Apr 9, 2003 1:18:47 PM EDT
Edit Web Application Deployment Descriptors...
Configuration
Targets
Deploy
Monitoring
Notes
General
Files
Other

Name: BEA_SWIFT_1_0_EventRouter.war

Path: C:\bea\user_projects\wlidomain\BEA_SWIFT_1_0_70.ear\BEA_SWIFT_1_0_E

Deployment Order: 1000

This pane enables you to select the trace level for the BEA WebLogic Integration framework.

For maximum tracing, enter DEBUG. This is recommended to obtain optimum debugging information for BEA support personnel.

The following levels are valid:

Table 1-3

Level	Indicates
AUDIT	An extremely important log message related to the business processing performed by an adapter. Messages with this priority are always written to the log.
ERROR	An error in the adapter. Error messages are internationalized and localized for the user.
WARN	A situation that is not an error, but that could cause problems in the adapter. Warning messages are internationalized and localized for the user.
INFO	An informational message that is internationalized and localized for the user.
DEBUG	A debug message, that is, information used to determine how the internal components work. Debug messages usually are not internationalized.

9. Click Apply to save the newly entered trace level.
10. Click BEA_*SWIFT*_1_0 EventRouter.
11. Click Persist to apply the logging changes.

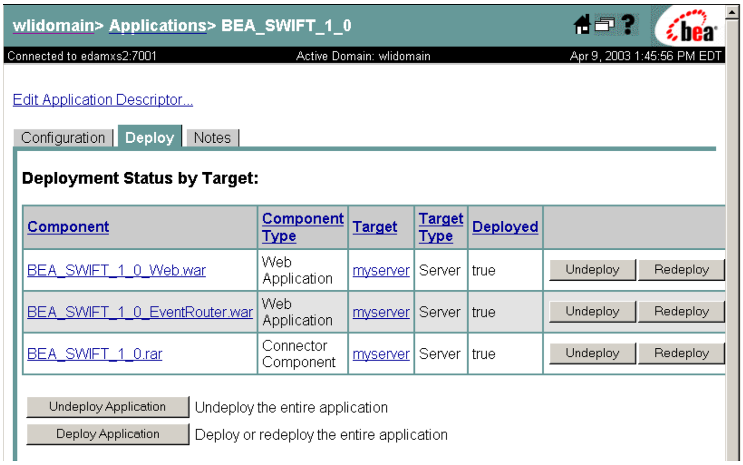
    This change need only be made once.

    It is set for all events associated with a given adapter.
12. Return to the WebLogic Server Console.
13. Select Applications from the WebLogic Server Console.
14. Select the adapter whose EventRouter you have modified in the previous steps.
15. Select the Deploy tab in the right pane.

    The right pane displays the following adapter components:

    - BEA_SWIFT _1_0.rar
    - BEA_SWIFT_1_0.web.rar
    - BEA_SWIFT_1_0_EventRouter.war.

Figure 1-5 WebLogic Server Console: Redeploy



wlidomain> Applications> BEA_SWIFT_1_0

Connected to edamxs2:7001 Active Domain: wlidomain Apr 9, 2003 1:45:56 PM EDT

Edit Application Descriptor...

Configuration | Deploy | Notes

Deployment Status by Target:

Component	Component Type	Target	Target Type	Deployed	
BEA_SWIFT_1_0_Web.war	Web Application	myserver	Server	true	Undeploy Redeploy
BEA_SWIFT_1_0_EventRouter.war	Web Application	myserver	Server	true	Undeploy Redeploy
BEA_SWIFT_1_0.rar	Connector Component	myserver	Server	true	Undeploy Redeploy

Undeploy Application — Undeploy the entire application

Deploy Application — Deploy or redeploy the entire application

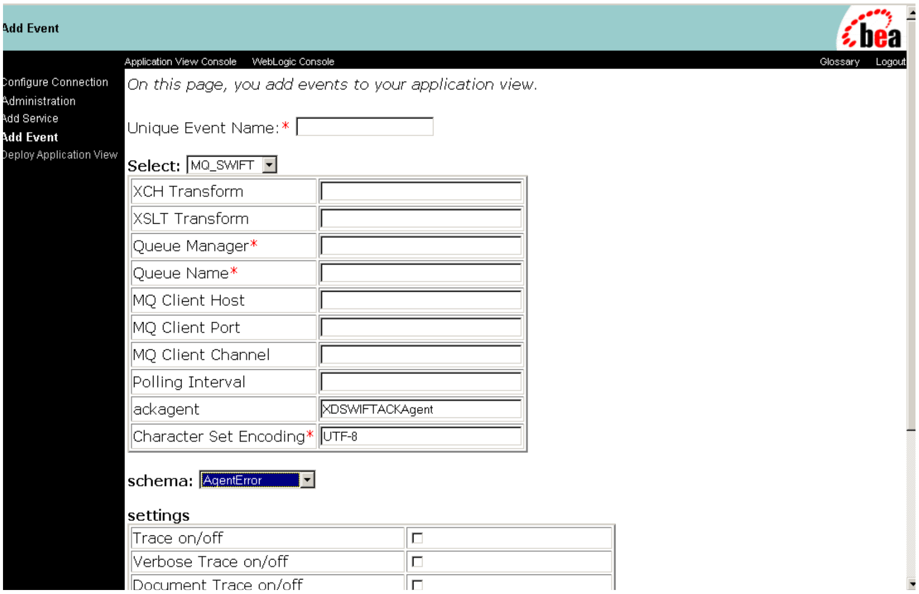
16. Redeploy the EventRouter by clicking the Redeploy button to the right of BEA_SWIFT_1_0_EventRouter.war.

Creating Adapter Logs for an Event

To create adapter logs for an event:

1. Create or modify the event.
2. Ensure that all of the adapter parameters are entered correctly.

Figure 1-6 Add Event Window



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

Unique Event Name:*

Select: MQ_SWIFT

XCH Transform	
XSLT Transform	
Queue Manager*	
Queue Name*	
MQ Client Host	
MQ Client Port	
MQ Client Channel	
Polling Interval	
ackagent	XDSWIFTACKAgent
Character Set Encoding*	UTF-8

schema: AgentError

settings

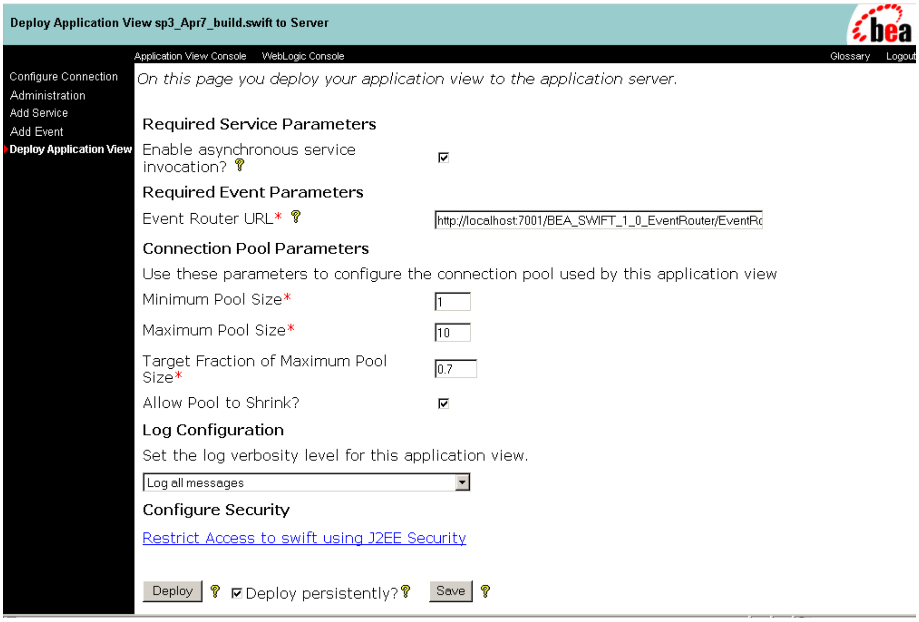
Trace on/off	☐
Verbose Trace on/off	☐
Document Trace on/off	☐

3. Select the appropriate schema from the drop-down list.
4. Select the appropriate trace levels as described in Table 1-2: Trace, Verbose trace, and Document trace.
5. Click Add to continue to the next configuration pane.
6. Click Continue to move to the next configuration pane.

   The Deploy Application View window opens.
7. Navigate to the Log Configuration area and select the desired trace level.

   This pane enables you to select the trace level for the BEA WebLogic Integration framework.

Figure 1-7 Deploy Application View Window



On this page you deploy your application view to the application server.

Required Service Parameters

Enable asynchronous service invocation? ☑

Required Event Parameters

Event Router URL* http://localhost:7001/BEA_SWIFT_1_0_EventRouter/EventRo

Connection Pool Parameters

Use these parameters to configure the connection pool used by this application view

Minimum Pool Size*	1
Maximum Pool Size*	10
Target Fraction of Maximum Pool Size*	0.7
Allow Pool to Shrink?	☑

Log Configuration

Set the log verbosity level for this application view.

Log all messages

Configure Security

Restrict Access to swift using J2EE Security

For maximum tracing, select Log all Messages. This is recommended to obtain optimum debugging information for BEA support personnel.

8. Click Deploy (or Save) to set the trace settings and deploy the application view.

   Traces are created the next time the event occurs.

   Traces are output to a file named BEA_SWIFT_1_0.log in the WebLogic Domain home directory.

APPENDIX

A Rules System Adapter

Document validation enables any document to be validated against sets of rules specified on a per-document basis. The rules are encoded in a rules file that is addressed through the system document dictionary.

Rules apply to document nodes in the XML tree, and (optionally) to their children. Built-in rules can be specified in any combination, and specialized rules can be coded in Java and loaded by the engine, as required.

BEA provides a complete set of built-in rules as required to validate SWIFT documents. The last part of this document describes how to write rules in Java for special situations. The following topics discuss how to encode a rules file, how to use the built-in rules, and how to code specialized rules:

- Rules File
- General Rule Set
- SWIFT Specific Rule Set
- Writing Rules in Java
- Writing Rule Search Routines in Java

Rules File

The rules file is an XML document. One file should exist for each document to be validated. The outer tag should be the document name and under this tag are rule tags, which may be enclosed within USING tags. For example, for an SWIFTMT540, the file (reduced considerably) might look like this:

```
<SWIFTMT540>
    <BASIC>
        <APPID_Application_ID_>F</APPID_Application_ID_>
...
        </INFO>
    </TRAILER>
</SWIFTMT540>
```

The rules document is an XML document tied into the BEA WebLogic Adapter for SWIFT through <validation> tags, which associate one or more rule documents with the specific document entry. The outer tag of the rules document should be explanatory, describing the type of document, but the actual tag is ignored. The rule document itself is a structure containing specific rule applications.

All attributes of rules must be specified in lowercase.

The document entry is the outer tag of the rules file. The name, document, is arbitrary and may be replaced with any meaningful XML-legal name.

Table 1-4 <*document*> entry

Attribute	Use
ackagent	The Java program class called to construct the acknowledgement. For the BEA WebLogic Adapter for SWIFT, this class is the XDSWIFTACKAgent.class. The actual agent to be used is selected in a search order: 1. The document specification in the repository. 2. This attribute of the outer tag of the rule file. 3. The listener configuration. As soon as an ackagent is located, it is selected for use, and the search ends.

Table 1-5 <using> entry

Attribute	Use
class	The Java program class contains all <rule>s within the section, unless overridden by a class=attribute in the <rule> entry itself.
other	Any unrecognized attribute is passed to each rule in the rule's attributes. For example, to apply the radix=',' attribute to all rules, specify it here rather than on each rule. Rules that do not use the radix attribute ignore it.

The rule tag and method attributes are required. The remaining attributes are rule-specific, and their inclusion is based on the rule itself. The validator uses the required tags to identify the rule in question and to identify the node or nodes of the document to which it applies. Common <rule> tags are:

Table 1-6 <rule> entry

Attribute	Use
tag	Names the right-most parts of the tag to which this rule applies. The rule applies to any node of the document that meets the tag criteria. For example, Document Table Model (DTM) would cause this rule to be applied to every DTM in the incoming document. X.DTM applies to all DTM parts prefixed by X. Tags are case sensitive. If omitted, a stag must be used.
stag	This is a specification subsection tag.
name	The rule's identification, which should be a unique name. This is used in trace messages to specify which rule caused a violation. If omitted, no unique identification can be given.
class	The rule class to which this rule belongs. This corresponds to a Java object class, and each rule is a method of the class. If this is omitted, the class from the enclosing USING tag is used.
method	The specific rule.
usage	Specify usage=M (mandatory) to check that there is a value in the identified node. This check is applied before the actual rule logic is executed.

The rule document is located by the <validation> tag value in the dictionary's system section and is identified with the specific document in its <document> entry. Rules validation is performed for document input (<in_validation>) and/or output (<out_validation>). If several tags are found in the document description, each rules validation is performed in the order in which the tags are found.

A section of the dictionary that illustrates this follows.

```
<system>
     <validation package="SWIFT">
         <name package="SWIFT"
file="templates/SWIFT/rules/SWIFTMT100rules.xml">100RULES</name>
     </validation>
</system>
   <document> package="SWIFT">SWIFTMT100
```

```
  <in_validation>100RULES</in_validation>
</document>
```

General Rule Set

The engine provides general rules for use by any rule set. The rules are located in com.ibi.edaqm.XDRuleBase, which extends com.ibi.edaqm.XDRuleClass, the base of all rules. Your own rule class should extend XDRuleBase instead of EXRuleClass. The general rules include:

- isN
- isR
- isDate
- isTime

isN

isN validates that a node is numeric with an optional leading sign.

```
<rule tag="xx" method="isN” />
```

Table 1-7 isN

Attribute	Meaning
Min	Minimum number of digits required, not including sign. Optional.
Max	Maximum number of digits permitted, not including sign. Optional.

isR

isR validates that a node is numeric with an optional leading sign and a single decimal point.

```
<rule tag="xx" class="XDSWIFTRules" method="isR" />
```

Table 1-8

Attribute	Meaning
Min	Minimum number of digits required, not including sign or radix. Optional.
Max	Maximum number of digits permitted, not including sign or radix. Optional.
Radix	A single character to be used to separate the decimal parts of the real value. The default is a period character. The radix attribute can be taken from the USING entry.

isDate

isDate validates that a node is in CCYYMMDD format.

```
<rule tag="xx" class="XDSWIFTRules" method="isDate" />
```

Table 1-9 isDate

Attribute	Meaning
Min	Minimum number of positions required. If omitted, 8 is assumed.
Max	Maximum positions permitted. If omitted, 8 is assumed.

isTime

isTime validates that a node is in HHMM[SS] format.

```
<rule tag="xx" method="isTime" />
```

Table 1-10 isTime

Attribute	Meaning
None	None.

SWIFT Specific Rule Set

The rules that are specific to SWIFT include:

- isValidReference
- isValidISIN
- isNotPresent
- isValidMultiLine
- isSWIFTReal
- isSWIFTDate
- isValidSWIFTString
- isSWIFTTime
- isValidMessageType
- checkValue
- checkCD
- checkRepetitive
- checkNodes
- checkChildSequence
- checkAddition
- checkRelation
- checkSegment

isValidReference

isValidReference validates the value with the following validations:

- The first character should not be /.
- The last character should not be /.
- At any place no two / should come together.

```
<rule tag="(node to check)" method="isValidReference"
errorcode="(error code)"/>
```

isValidISIN

isValidISIN validates the value with the following validations:

- The first four characters should be ISIN.
- The fifth character should be a space.
- The maximum characters should be 17 including the ISIN and the space that follows it.

```
<rule tag="(node to check)" method="isValidISIN" errorcode="(error
code)"/>
```

Warning: The format of the ISIN given in the input is valid. The validity of the ISIN can be checked in the SWIFT ISIN directory.

isNotPresent

IsNotPresent validates whether or not the value is present.

```
<rule tag="(node to check)" method="isNotPresent" errorcode="(error
code)"/>
```

isValidMultiLine

isValidMultiLine validates whether or not the value is present.

The value should be alphanumeric.

The line ends with a carriage return and a new line character.

```
<rule tag="(node to check)" method="isMultiLine" line=”3” min=”2”
max=”10” errorcode=”(error code)”/>
```

Table 1-11 isValidMultiLine

Attribute	Meaning
Line	Number of valid lines. The minimum number of lines is 1.
Min	Minimum number of characters.
Max	Maximum number of characters.
Errorcode	The error code for the rule.

isSWIFTReal

isSWIFTReal validates the value with the following validations:

- The value is checked for Real numbers with at least one character before the decimal point (the decimal point is “,”).
- The value is checked for having a mandatory decimal point.

  Note: The decimal comma is included in the maximum length.

```
<rule tag="(node to check)" method="isSWIFTReal" min=”2” max=”10”
errorcode=”(error code)”/>
```

Table 1-12 isSWIFTReal

Attribute	Meaning
Min	Minimum number of characters.
Max	Maximum number of characters.
Errorcode	The error code for the rule.

isSWIFTDate

isSWIFTDate validates the date according to the format specified in the attribute. The following are valid formats:

- date1: MMDD
- date2: YYMMDD
- date3: YYMM
- date4: YYYYMMDD
- date5: date4 + value date (the year should be between 1980 and 2060)

```
<rule tag="node to check" method="isSWIFTDate" format=”date1” />
```

Table 1-13 isSWIFTDate

Attribute	Meaning
Format	The format of the date to check against.
Errorcode	The error code for the rule.

IsValidSWIFTString

isValidSWIFTString validates the value according to the format specified.

The valid values of the format are:

x	The S.W.I.F.T X character list.
y	The S.W.I.F.T Y character list.
z	The S.W.I.F.T Z character list.
c	Alphanumeric capital letters and digits only.

a

Alphabetic capital letters only.

h

Hexadecimal characters only.

e

Blank spaces.

Exceptions to this rule are:

- A value cannot have blank spaces alone.
- A value cannot have CrLf characters alone.

This function validates each and every digit and/or character against the standard SWIFT recognized X character list.

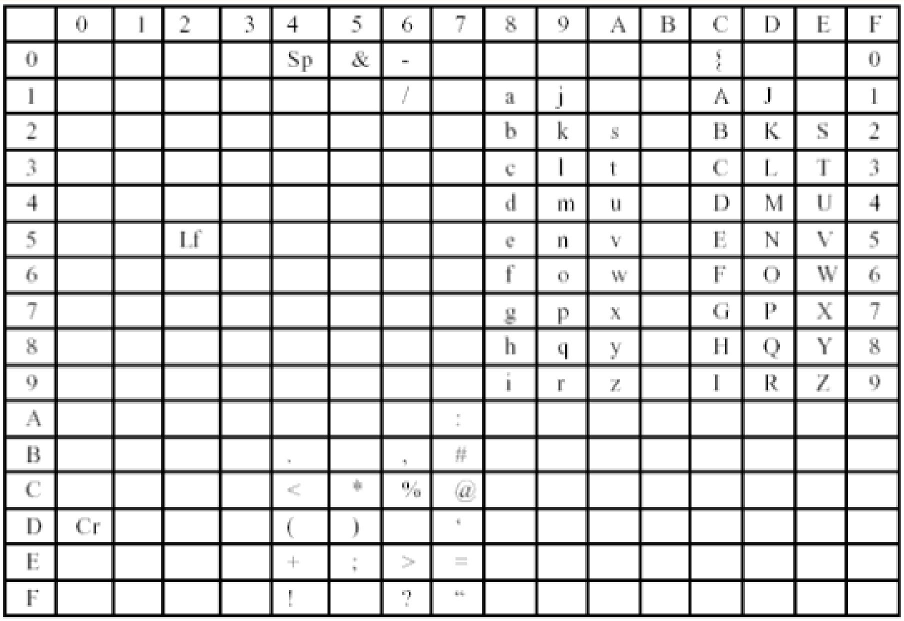
SWIFT X Character Set
A to Z (uppercase) a to z (lowercase)
0 to 9
/ (forward slash), - (minus sign), ? (question mark), : (colon), ((left parenthesis),) (right parenthesis), . (point), , (comma), ’(right single quote), + (plus sign), SPACE, CrLf (line feed, new line, and carriage return characters)

SWIFT Y Character Set
A to Z (uppercase)
0 to 9
/ (forward slash), - (minus sign), ? (question mark), : (colon), ((left parenthesis),) (right parenthesis), . (point), (comma), ’(right single quote), + (plus sign), SPACE, = (equal to)
! (exclamation mark), ” (right quotes), % (percentage), & (ampersand), * (asterisk), ; (semi-colon), < (left V bracket), >(right V bracket)

SWIFT Z Character Set
A to Z (uppercase) a to z (lowercase)
0 to 9
/ (forward slash), - (minus sign), ? (question mark), : (colon), ((left parenthesis),) (right parenthesis), . (point), , (comma), ’ (right single quote), + (plus sign), SPACE, Cr Lf (line feed, new line, and carriage return), = (equal to), @ ,#, { (left brace)
! (exclamation mark),” (right quotes), % (percentage), & (ampersand), * (asterisk), ;(semi-colon), < (left V bracket), >(right V bracket)

Hexadecimal Representation of SWIFT Character Set

SWIFT characters can be comprised of hexadecimal characters. These representations are different from regular IBM hexadecimal representations of characters.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0					Sp	&	-						{			0
1							/		a	j			A	J		1
2									b	k	s		B	K	S	2
3									c	l	t		C	L	T	3
4									d	m	u		D	M	U	4
5			Lf						e	n	v		E	N	V	5
6									f	o	w		F	O	W	6
7									g	p	x		G	P	X	7
8									h	q	y		H	Q	Y	8
9									i	r	z		I	R	Z	9
A								:								
B					.		,	#								
C					<	*	%	@								
D	Cr				(	)		‘								
E					+	;	>	=								
F					!		?	“								

For example, an Lf character can be depicted as 25, an.

& can be depicted as 50 and an ! can be depicted as 4f.

```
<rule tag="(node to check)" method="isValidSWIFTString" type=”x”
errorcode=”(error code)”/>
```

Table 1-14 isValidSWIFTString

Attribute	Meaning
Type	The type of string format to check against.
Min	Minimum number of characters.
Max	Maximum number of characters.
Errorcode	The error code for the rule.

isSWIFTTime

isSWIFTTime validates the value according to the format specified.

The valid values of the format are:

- time1: HHMM.
- time2: HHMMSS.
- time3: HHMMSSsss (where *sss* stands for microseconds).

```
<rule tag="(node to check)" method="isValidSWIFTTime"
format=”time1” errorcode=”(error code)”/>
```

Table 1-15 isSWIFTTime

Attribute	Meaning
Format	The type of date format to check against.
Errorcode	The error code for the rule.

isValidMessageType

isValidMessage Type validates the message type value.

The message type value should be greater than 100 and less than 999.

There is an exception if the message is one of the following sets:

101,102,204,206,207,256,303,304,405,416,503,504,505,506,507,527,569, or 575.

A warning message indicating that sending or receiving one of the above messages requires a MUG (ISO 7775 Message User Group) registration and that special validation is required as per individual MUG standards is sent to the console and/or logged in the trace file.

All SWIFT Adapter users are enrolled in MUG.

```
<rule tag="(node to check)" method="isValidMessageType"
errorcode=”(error code)”/>
```

Table 1-16 isValidMessageType

Attribute	Meaning
Errorcode	The error code for the rule.

You can learn more about the ISO 7775 MUG at:

```
http://www.swift.com/index.cfm?item_id=41974
```

checkValue

checkValue validates the elements or components of a segment for presence according to a pattern. The patterns validate the code in the independent variable. The tag= and tagset= attribute can be used to locate the section to which the rule applies.

Table 1-17 CheckValue

Attribute	Meaning
Tagset	List of nodes to check against.

Table 1-17 CheckValue

Attribute	Meaning
Code	The expression used to validate the elements.
Errorcode	The error code for the rule.

Case 1:

```
<rule tag="Parent of A and B" method="checkValue" tagset="A,B"
code="1=@reasonList/2=@statusList" errorcode="(error code)"/>
```

If the Value of A is one of the code set ‘reasonList,’ then the value of B should be one of the code set of statusList.

Case 2:

```
<rule tag="Parent of A and B" method="checkValue" tagset="A,B"
code="1=x;y;z/2=a;b;c" errorcode="(error code)" />
```

If the Value of A is one of the set {x,y,z}, then Value of B should be one of the set {a, b, c}.

Case 3:

```
<rule tag="Parent of A and B" method="checkValue" tagset="A,B"
code="1!?/2=?" errorcode="(error code)" />
```

If element A is not present, element B should be present.

Case 4:

```
<rule tag="Parent of A,B" method="checkValue" tagset="A,B"
code="1=?/2=?,1!?/2!?" errorcode="(error code)"/>
```

Either A and B should be present, or A and B should not be present.

Case 5:

```
<rule tag="Parent of A,B, C" method="checkValue" tagset="A,B,C"
code="1=?/2!?+3!?,1!?/2=?+3=?" errorcode="(error code)"/>
```

Either element A or (B and C) must be present.

Case 6:

```
<rule tag ="Parent of x,y,z,A,B" method="checkValue"
tagset="x,y,z,A,B" code="1=?|2=?|3=?/4=?+5=? errorcode="(error
code)"/>
```

If any of the elements x, y, and z are present, then element A and element B must be present.

Case 7:

```
<rule tag = "Parent of A, X, Y, Z, x, y, z, …" method = "checkValue"
tagset = "A, X, Y, Z, x, y, z, …" Code = "1=XXX +
(2=?|3=?|4=?…)/5=?|6=?|7=?… errorcode="(error code)"/>
```

If element A contains the word XXX and any of the elements {X,Y,Z,X1,Y1,Z1,…} are present, then any of the elements {x,y,z,x1,y1,z1….} should be present.

checkCD

checkCD validates that a node has appropriate sub nodes. The valid codes and definitions of the codes are as follows:

Table 1-18 checkCD Valid Codes

Condition Code	Meaning	Definition
R	Required	At least one of the elements in the condition must be present.
E	Exclusion	Not more than one of the elements specified in the condition can be present.
G	Repetitive Sequence Related	The first element must be present if there are no repetitive sequences of the second or third element.
F	Repetitive Sequence Related	The first element must be present if there are repetitive sequences of the second element.
V	Multiple Occurrences Related	If the first element is present and the second element is present at least once, then the value of all the occurrences of the second element should be equal to the first element.

Table 1-18 checkCD Valid Codes

Condition Code	Meaning	Definition
M	Mandatory	One of the children of the specified elements must contain a value.

Table 1-19 checkCD

Attribute	Meaning
Tagset	List of nodes to check against.
Cd	The expression used to validate the elements.
Errorcode	The error code for the rule.

Case 1:

```
<rule tag=”Parent of A and B” method=”checkCD” tagset=”A,B”
cd=”E0102” errorcode=”(error code)”/>
```

Mutually exclusive A and B.

Case 2:

```
<rule tag=”Parent of A,B” method=”checkCD” tagset=”A,B” cd=”R0102”
errorcode=”(error code)”/>
```

Either element A or element B or both must be present.

Case 3:

```
<rule tag=”Parent of A,B, C” method=”checkCD” tagset=”A,B,C”
cd=”G010203” errorcode=”(error code)”/>
```

Element A must be present if there are no repetitive sequences B or C (this is applicable to MT 573).

Case 4:

```
<rule tag=”Parent of A,B” method=”checkCD” tagset=”A,B” cd=”F0102”
errorcode=”(error code)”/>
```

Element A must be present if there are repetitive sequences of element B (two or more times).

Case 5:

```
<rule tag=”Parent of A,B” method=”checkCD” tagset=”A,B” cd=”V0102”
errorcode=”(error code)” />
```

If element A is present and there is at least one element B, then the value of all occurrences of element B should be equal to the value of element A.

Case 6:

```
<rule tag=”Parent of A,B” method=”checkCD” tagset=”A,B” cd=”M0102”
errorcode=”(error code)” />
```

One of the children from element A and element B must contain a value.

checkRepetitive

checkRepetitive validates that the value of the specified element should be the same in all occurrences if the specified element is used or present repetitively.

```
<rule tag=”(Root of the document)” search=”A” errorcode=”(error
code)”/>
```

Table 1-20 checkRepetitive

Attribute	Meaning
Search	The node to search for and check.
Errorcode	The error code for the rule.

Note: A is the name of the actual node (not a fully qualified name). It is specified the root of the document, instead of the parent of A, because this rule searches the entire document for A.

checkNodes

checkNodes validates either one or more of elements from a set A, A1, … of nodes must be present or one or more of the set B, B1, … of nodes must be present, but not elements from both sets. For example, {A, A1, A2, …} and {B, B1, B2, …} are two sets of nodes defined. Only nodes from one of the sets can exist. If node A and node B exists, then the rule fails.

```
<rule tag="Parent of A,A1,…B,B1,…" method="checkNodes"
tagset="A,A1,A2,A3,…;B,B1,B2,B3……" errorcode="(error code)" />
```

Table 1-21 checkNodes

Attribute	Meaning
Tagset	The two sets of nodes to check separated by ";".
Errorcode	The error code for the rule.

checkChildSequence

checkChildSequence checks for a specified node for its occurrences, and the presence of other nodes is based upon it.

Table 1-22 checkChildSequence

Attribute	Meaning
Start	Node to check if it is repetitive.
Pattern	The expression used to validate the elements.
Errorcode	The error code for the rule.

Case 1:

```
<rule tag="Parent of A" method="checkChildSequence" start = A
pattern= B errorcode="(error code)" />
```

If tag A is repetitive and is present two or more times, then tag B should be one of the children of tag A.

Case 2:

```
<rule tag=”Parent of A” method=”checkChildSequence” start = “A”
pattern= “B1,B2,B3….” errorcode=”(error code)” />
```

If tag A is repetitive and is present two or more times, then one of the tags {B1,B2,B3,….} must be present.

checkAddition

checkAddition validates the value of a specified element to be equal to the sum of all values of another element.

Table 1-23 checkAddition

Attribute	Meaning
Tagset	List of nodes to check.
Errorcode	The error code for the rule.

Case 1:

```
<rule tag=”Parent of A, B, C” method=”checkAddition” tagset=”A,B,C”
errorcode=”(error code)” />
```

The value of element A should be equal to the sum of all values of element B (they are repetitive optionally) or the sum of all values of element C (they are repetitive optionally).

The validation applies only if the element A exists in the incoming SWIFT message. If the element A exists, then at least one of the elements of B should exist or at least one of the elements of C should exist.

Case 2:

```
<rule tag=”Parent of A, B” method=”checkAddition” tagset=”A,B,C”
errorcode=”(error code)”   />
```

The value of element A should be equal to the sum of all values of element B (they are repetitive optionally).

The validation applies only if the element A exists in the incoming message. If the element A exists, then at least one element B should exist.

checkRelation

checkRelation validates whether element A or one or more of the set {x,y,z,…} is present, then element B should be present and must be succeeding all occurrences of A or one or more of the set {x,y,z…}. The converse is also true.

```
<rule tag=”Parent of A,B,x,y,z,.,.,” method=”checkRelation”
tagset=”A” taglist=”x,y,z” find=”B” errorcode=”(error code)”   />
```

Table 1-24 checkRelation

Attribute	Meaning
Tagset	Node to check.
Taglist	List of nodes to check.
Find	Node to find to see if it should be present.
Errorcode	The error code for the rule.

checkSegment

Segment D is mandatory when in any occurrence of segment C, sub-segment C1 is present, and the sub-segment C1a is not present.

Note: C1a is the child of C1. When specified in the rule, specify the node with its full name.

```
<rule tag="Parent of C,C1,C1a,D" method="checkSegment" parent="C"
subseq="C.C1" child="C.C1.C1a" check="D" errorcode="(error code)"
/>
```

Table 1-25 checkSegment

Attribute	Meaning
Parent	The parent node.
Subseq	The sub-segment node.
Child	The child node.
Check	The node to check if present.
Errorcode	The error code for the rule.

Writing Rules in Java

Rules can be written in Java, loaded by the system at startup, and applied by specification in a rule. A rule class extends XDRuleClass and can make use of any of its services. Each public method in the rule class that meets the rule signature can be applied by name as a rule. The rule methods can make use of service methods in the parental XDRuleClass.

In this example, a node is checked to determine whether its value is the word identified by the value= attribute. If not, it is an error.

The following parameters are passed:

Node	The node identified by the tag attribute in the rule. The rule method will be called once for each node that matches the tag specification.
Value	The data value of the addressed node. This differs from the node.getValue() return if the tag contained a subfield address (for example, tag=x:2).

Attributes	A HashMap of rule attributes. The rule method can check for any attributes that it requires. A HashMap is a fast implementation of a Hashtable that does not serialize.

Listing A-1 Node Checking Example

```
import java.util.*;
import com.ibi.edaqm.*;
public class XDMyRules extends XDRuleClass
{
    public XDMyRules()
    {
  }
    public void specialRule(XDNode node, String value,
                         HashMap attributes)
                throws XDException
    {
        trace(XD.TRACE_DEBUG, "specialRule called with parms: " +

              node.getFullName() + ", " + attributes.toString());
        String testValue = (String)attributes.get("value");
        if (value.equals(testValue))
        {
 node.setAssociatedVector(new XDEDIError(4, 0,
error,"explanation"));
           throw new XDException(XD.RULE, XD.RULE_VIOLATION,"node
value
"+value+" is not 'Value'");

      }
  }
}
```

Rule violations should throw an XDException describing the violation.

The parental class provides a group of services to assist in preparing rules:

Table 1-26 Services for Preparing Rules

Method	Purpose
Boolean is YYYYMMDD (string date).	Validates that a date is formatted correctly.

Table 1-26 Services for Preparing Rules

Method	Purpose
Boolean is InList (string list, string value).	The value must be in the list.
Void trace (int level, string msg).	The text of the message is written to the system trace file. The level should be XD.TRACE_DEBUG. XD.TRACE_ERROR or XD.TRACE_ALL.

Rules can also use all methods in XDNode to address the values in the passed node and the tree in general.

Rule violations must be returned as XDExceptions of class XD.RULE. Two causes are available, XD.RULE_SYNTAX if the rule is in error, and XD.RULE_VIOLATION if the data violates the rule. Syntax errors cause the document to be aborted, as it is presumed that rules should have been debugged. Violations should be posted to the node by the rule, and the engine continues to process the document. Violations are traced by the engine and affect the later acknowledgement generation.

The error itself is posted to the node by the standard XDNode service setAssociatedVector(Object o) which records an object with the node. The special EDIError object contains the elements:

Class	Class of the error. Should be 4 for a syntax error, resulting in an AK4.
Reserved	Must be 0.
Error code	Code to be returned in the ack AKx (997).
Explanation	A string explaining the error, for tracing use.

Writing Rule Search Routines in Java

Short lists can be searched by built-in rule engine code. Longer lists, in which the values in the list are obtained not from the attribute directly but instead from an external source, require a rule list searcher tailored to the source. Lists might be obtained from:

- A simple file.
- A database with values loaded at startup.
- A database with an access at each search request.

Each list might require its own search logic, tailored to the source and format of the list itself. To accommodate this, the rule engine allows list-specific search routines to be developed and added to the system. These routines are loaded at system initialization and terminated at system closedown. Each must offer a search method that determines whether the passed value is valid.

Search routines must extend the XDRuleList class that is part of the edaqm package: com.ibi.edaqm.XDRuleClass. The routine must offer these methods in the manner common to all XD extensions:

- init(String args) is called once at system initialization.
- term() is called once at system termination. It is not guaranteed that it is called.
- search(String value) is called when the rule is executed.

The Rule List search code is identified in the <preload> section of the <system> area of the dictionary. The Preloads console page manages this section.

```
<preload>
  <name file="RuleFileList(c:\ziplist.txt)" comment="validates zip 
codes">ziplist</name>
</preload>
```

The rule tag specifies that a rule can be written:

```
<rule tag=”xxx” code=”@ziplist” method=”checklist”/>
```

that names the preloaded routine. This routine might load a list from a text file. A simplified example procedure to load a file containing codes follows.

Listing A-2 Loading a File Example

```
import com.ibi.edaqm.*;
import java.util.*;
import java.io.*;
/**
 * A rule list handler is a routine called to enable users search 
lists during execution
```

```
 * or the checkList rule. checkList() is a generally available rule
to test whether the
 * contents of a document field are valid. The rule list handler is
invoked when
 * the code= attribute indicates the name of a coder routine rather
than a simple list.<P>
 * For example, <I>code="@list1"</I> will cause the search routine
of the list1 class to
 * be invoked.<P>
 * The file read by this procedure consists of tokens separated by
new line, white space or commas.
 */
public class XDRuleListFile extends XDRuleList
{
        String[] list;
        ArrayList al = new ArrayList(127);
        public XDRuleListFile()
        {
      }
        /**
         * The init method is called when a rule is loaded. It can
perform any necessary
        * initialization, and can store any persistent information
in the object store.
         *
        * @param parms Array of parameter string passed within the
start command init-name(parms).
         */
        public void init(String[] parms) throws XDException
        {
            if (parms == null)
            {
               throw new XDException(XD.RULE, XD.RULE_SYNTAX, "no
parms sent to " + name);

          }

            try
            {
               File f = new File(parms[0]);
                FileInputStream fs = new FileInputStream(f);
                long len = f.length();
                byte[] b = new byte[(int)len];
                fs.read(b);
                fs.close();
                String data = new String(b);
             StringTokenizer st = new StringTokenizer(data, ", " +
XD.NEWLINE);
```

```
                  while (st.hasMoreTokens())
                  {
                      String part = st.nextToken();
                      al.add(part);
                  }
              }
                catch (FileNotFoundException e)
                {
                  throw new XDException(XD.RULE, XD.RULE_SYNTAX, "list
file
"+parms[0] + " not found");

          }
            catch (IOException eio)
            {
                throw new XDException(XD.RULE, XD.RULE_SYNTAX,
eio.toString());
          }
      }
        /**
       * The term() method is called when the worker is terminated.
It is NOT guaranteed
         * to be call, and applications should not rely upon this
method to update data bases or
         * perform other critical operations.
         */
        public void term()
        {
      }

        /**
        * Search the given value to determine whether it is in the
list.
         *
         * @param value String to test against the list
         * @return true if found, false otherwise
         */
        public boolean search(String value)
        {
            return al.contains(value);
      }
}
```

B Linking SWIFT Adapters to SWIFT Networks

This section describes the connecting of business applications to SWIFTAlliance and includes the following topics:

- Batch File Transfer – FILE and FTP
- Application Server – CAS MF
- Interactive – MQ Series

SWIFT allows the connecting of business applications to SWIFTAlliance which can be done in different ways.

- **Manual File Transfer**: S.W.I.F.T. messages are exchanged between the business application and SWIFTAlliance in batch files *with* operator intervention on (either and/or both) the business application and the SWIFTAlliance.
- **Automated File Transfer**: S.W.I.F.T. messages are exchanged between the business application and SWIFTAlliance in batch files, **without operator intervention**. For example, the file transfer operation is automated on (either or/and both) SWIFTAlliance and Business Application.
- **Interactive**: Unlike file transfer operation mode, S.W.I.F.T. messages are exchanged real time (using a conversational protocol) between the business application and SWIFTAlliance on an individual basis and **without operator intervention** on both SWIFTAlliance and Business Application.

The BEA WebLogic Adapter for SWIFT supports all three options when used in conjunction with other BEA adapters such as FTP, File, TCP/IP, and MQ Series. The following table summarizes the connectivity options.

Mode	BEA Adapter	Third Party Software
File Transfer	BEA WebLogic Adapter for SWIFT BEA WebLogic Adapter for File	None specific but could be application, for example, MERVA or BESS
Application Server	BEA WebLogic Adapter for SWIFT BEA TCP/IP Adapter	'AI MXA TCP-IP' CASmf Application Server
Interactive	BEA WebLogic Adapter for SWIFT BEA WebLogic Adapter for MQSeries	MQSA SWIFT Alliance Toolkit Runtime

Batch File Transfer – FILE and FTP

The File Transfer method permits batch file transfer with message partners.This method permits both automated and manual invocation of communication sessions, either with or without the use of parameter files. For each message format, the communication media may be diskettes or files, that is, read or write a batch message file from, or to a directory in a local or remote file system.

The following message file formats are supported:

- **DOS-PCC** is used for batch input and output of messages. The DOS-PCC connection method permits you to read or write an ST200 DOS message file.
- **RJE** (Remote Job Entry) is used for batch input and output of messages in ST200 RJE format.
- **MERVA/2** batch transfer (from/to mainframes) in IBM MERVA/2 format.

- **CAS** (Common Application Server) format.

Application Server – CAS MF

The SWIFT Common Application Server (CAS) defines how data can be exchanged between SWIFTAlliance and financial applications through a conversational protocol. The specifications are common to both SWIFTAlliance and ST400, thus allowing smooth ST400 migration to SWIFTAlliance.

CAS supports TCP/IP and SNA LU6.2 as a networking protocol. Application developers need not know the CAS protocol. The CASmf software package provides APIs to the financial application developers. It hides all communication and data formatting aspects, enabling developers to concentrate on the application functions. APIs exist to open, close, or abort a session and send or receive data.

CASmf is available on the following platforms:

- AIX
- HP/UX
- Sun OS
- Windows NT
- AS400
- Open VMS for VAX and Alpha

CASmf uses TCP/IP as the communication protocol. It can run on the same system as SWIFTAlliance. CASmf can handle several simultaneous application sessions. The BEA WebLogic Adapter for SWIFT can use the TCP/IP Adapter for processing events and services between BEA WebLogic Integration and SWIFT CASmf applications. The option 'AI MXA TCP-IP' must be licensed on SWIFTAlliance.

Interactive – MQ Series

The MQSeries Interface for SWIFTAlliance (MQSA) software provides a reliable communication between financial applications and SWIFTAlliance through IBM MQSeries. It enables S.W.I.F.T. message exchange between SWIFTAlliance and financial applications. The MQSA software is based on the SWIFTAlliance Development Toolkit referenced as ADK in this document. It uses ADK functions to communicate with SWIFTAlliance and MQSeries functions to access message queuing services.

The MQSeries Interface for SWIFTAlliance is available for SWIFTAlliance Access running on Windows NT and UNIX. IBM MQSeries messaging software enables business applications to exchange information across different operating system platforms in a way that is straightforward and easy for programmers to implement.

MQSeries is available on different platforms (Windows NT, UNIX, OS/400, MVS/ESA, and so forth). The applications are shielded from the mechanics of the underlying communications. With MQSeries, the exchange of messages between the sending and receiving program is time independent. This means that the sending and receiving applications are de-coupled so that the sender can continue processing without having to wait for the receiver to acknowledge the receipt of the message.

The MQSA software enables fast integration of user applications with SWIFTAlliance Access. It is composed of two ADK components, and they must be installed as any other ADK component. The components must be registered in SWIFTAlliance. The purpose of the registration is to make the component known to SWIFTAlliance. The registration adds component data to the SWIFTAlliance database.

The MQSA software is limited to the exchange of S.W.I.F.T. messages, S.W.I.F.T. ack/nacks and recording of events in the SWIFTAlliance journal. It can handles multiple MQSeries queues for the connection with the user application(s).

The BEA WebLogic Adapter for SWIFT and BEA WebLogic Adapter for MQSeries provide connectivity to SWIFT network using SWIFT Alliance MQSA. The SWIFTAlliance Toolkit run-time license is required to run the MQSA software.

APPENDIX

C Support for AS1 and AS2 Communications

This section describes support for AS1 and AS2 communications and includes the following topic:

- Choosing AS1 (SMTP/e-mail) or AS2 (HTTPS)

AS1 and AS2 document exchange is an adapter messaging component that provides for the secure exchange of electronic documents with interchange partners via the Internet and VANs (Value Added Networks). A BEA adapter supplies matching interoperability support for EDIINT AS1 and AS2, as well as support for the underpinning technologies HTTPS and SMTP.

The BEA WebLogic Adapter for SWIFT enables you to send and receive interchange documents over the Internet through a variety of transfer protocols. This flexible service allows the use of in-house protocols, without the need to consider specific protocol support on the opposite end of the transaction. The Transformation Servers managed data and document exchange service is based on open standards, including matched interoperability support for EDIINT AS1 and AS2. This provides maximum flexibility for your interchange partners, as they may use either SMTP (S/MIME) protocol or HTTPS, regardless of source protocol.

Choosing AS1 (SMTP/e-mail) or AS2 (HTTPS)

AS1

- Asynchronous SMTP server connection with your interchange partner.

- Transaction-time based on email server reaction.
- Supports EDIINT AS1 compatible applications.
- Supports digital signatures.
- Non-repudiating.
- File size limited to email capacity.

AS2

- Synchronous communication.
- Supports digital signatures
- Non-repudiating
- Ease of integration with back-end systems.
- Supports EDIINT AS2 compatible applications.
- Unlike AS1, there is no externally imposed file-size limitation as the connection is direct and immediate.
- Persistent connectivity to the Internet.

AS1 and AS2 are comprised and assembled from existing standards to produce an organized and managed specification for the independent transport of interchange documents over the Internet. The philosophy behind the IETF Working Group is to provide an international transport mechanism whereby interchange partners may communicate over the open Internet, within a secure and supportable specification, without using paid framework services.

The specifications (AS1 & AS2) were developed in near parallel, and the consecutive naming convention does not reflect any statement of version or relationship, other than the fact that AS2 references AS1 in its specification. AS1 utilizes SMTP (Simple Mail Transfer Protocol), with S/MIME for encryption and security, by requiring authentication, message integrity, and originating non-repudiation. AS2 may be considered a modification of AS1, providing S/MIME via direct HTTP or HTTP/S as the transport protocol.

The AS2 specification provides support for “any” data-type transmission via the Internet over HTTP. The AS2 specification governs data transport, not specific data-type validation or document processing. The AS2 specification designates the means by which to connect, deliver, validate, and acknowledge transport in a secure reliable manner.

The BEA WebLogic Adapter for SWIFT can be used in conjunction with the following format
adapters to provide support for AS1- and AS2-based EDI communication:

- EDI ANSI X12
- EDI EDIFACT
- HIPAA
- HL7
- SWIFT