



BEA WebLogic Adapter for SWIFT®

User Guide

Copyright

Copyright © 2003 BEA Systems, Inc. All Rights Reserved.

Portions Copyright © 2003 iWay Software. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Contents

About This Document

Who Should Read This Documentation.	x
Additional Information	x
How to Use This Document.	xi
Contact Us!	xii
Documentation Conventions	xii

1. Introducing the BEA WebLogic Adapter for SWIFT

About the BEA WebLogic Adapter for SWIFT	1-1
Supported Features for Application Integration	1-2
Supported Services	1-2
Supported Events	1-3
Benefits of the Adapter for SWIFT	1-3
Components of the Adapter Kit	1-4
SWIFT Document Validation.	1-5
Validation of Inbound SWIFT Documents	1-5
Validation of Outbound SWIFT Documents	1-6
Post-Validation Acknowledgements	1-7
Getting Started With the Adapter for SWIFT	1-8
Step 1: Design the Application Integration Solution	1-8
Step 2: Determine the Required Business Processes for SWIFT Documents	1-9
Step 3: Generate Schemas and Define Document Transformations.	1-9

Step 4: Define Application Views and Configure Services and Events	1-10
Step 5: Define Validation for SWIFT Documents	1-10
Step 6: Integrate with Other BEA Software Components	1-10
Step 7: Deploy the Solution to the Production Environment.	1-11

2. Transforming Document Formats

About Schemas	2-1
Service Requests	2-2
Service Responses	2-2
Events.	2-2
About Document Format Transformations	2-2
Transforming SWIFT to XML (Events Only)	2-3
Transforming XML to SWIFT (Services Only).	2-3
About Schema Repositories	2-3
Contents of the Schema Repository.	2-4
About the Repository Manifest	2-5
Naming Schema Repositories	2-6
Modifying the Repository	2-7
Modify Repository on Disk	2-7
Modify Repository and Update the EAR File	2-8
Generating Transformation Templates and Document Schemas	2-8
About the Sample Utilities.	2-9
Extracting the Sample Utilities	2-9
Generating Transformation Templates.	2-9
Generating Document Schemas	2-10
Automatically Generating a Session Repository	2-10
Next Steps	2-10

3. Defining Application Views for SWIFT

How to Use This Document	3-2
Before You Begin	3-2
About Application Views	3-3
About Defining Application Views	3-3
Defining Service Connection Parameters	3-5
Setting Service Properties	3-6
MQ Service	3-7
File Service	3-8
FTP Service	3-9
Common Service and Event Settings	3-11
Setting Event Properties	3-12
MQ Event	3-13
File Event	3-15
FTP Event	3-17
TCP Event	3-19
Defining Event Connection Parameters	3-20
Testing Services	3-23
Testing Events Using a Service	3-26
Testing Events Manually	3-28

A. Validation Rules

About the Rules File	A-2
<document> tag	A-3
<using> Tag	A-3
<rule> tag	A-3
Writing Rules in Java	A-5
Writing Rule Search Routines in Java	A-8

General Validation Rules Reference	A-11
isN	A-11
isR	A-12
isDate	A-12
isTime.	A-13
SWIFT Specific Rules Reference	A-14
isValidReference	A-15
isValidISIN.	A-15
isNotPresent	A-15
isValidMultiLine	A-15
isSWIFTReal	A-16
isSWIFTDate	A-16
IsValidSWIFTString.	A-17
SWIFT X Character Set	A-18
SWIFT Y Character Set	A-18
SWIFT Z Character Set	A-18
Hexadecimal Representation of SWIFT Character Set	A-19
isSWIFTTime.	A-20
isValidMessageType.	A-20
checkValue	A-21
Case 1	A-21
Case 2	A-22
Case 3	A-22
Case 4	A-22
Case 5	A-22
Case 6	A-22
Case 7	A-22
checkCD.	A-23

Case 1	A-23
Case 2	A-24
Case 3	A-24
Case 4	A-24
Case 5	A-24
Case 6	A-24
checkRepetitive	A-24
checkNodes.	A-25
checkChildSequence.	A-25
Case 1	A-26
Case 2	A-26
checkAddition.	A-26
Case 1	A-26
Case 2	A-27
checkRelation	A-27
checkSegment.	A-28

B. Handling Acknowledgements

About Acknowledgement Processing	B-1
Processing Documents With Validation and Acknowledgement	B-2
About the Acknowledgement Agent	B-3
Acknowledgement Message Handling	B-4
Creating an Acknowledgement Event	B-5

C. Linking Business Applications to SWIFTAlliance

Connecting Business Applications to SWIFTAlliance	C-1
Connectivity Options.	C-2
Batch File Transfer – FILE and FTP	C-2

Application Server – CAS MF C-3
Interactive – MQ Series C-4

D. Adapter Support for AS1 and AS2 Communications

About the AS1 and AS2 Standards D-1
Comparison of AS1 (SMTP/e-mail) and AS2 (HTTPS). D-2
Adapter Support for AS1 and AS2 D-3

Index

About This Document

This document describes how to use the BEA WebLogic Adapter for SWIFT. This document is organized as follows:

- [Chapter 1, “Introducing the BEA WebLogic Adapter for SWIFT,”](#) describes the adapter and how it relates to both SWIFT documents and WebLogic Integration.
- [Chapter 2, “Transforming Document Formats,”](#) describes how to generate schemas and define document transformations for your SWIFT and XML documents.
- [Chapter 3, “Defining Application Views for SWIFT,”](#) describes application views and how to configure events and services.
- [Appendix A, “Validation Rules,”](#) describes how to encode a rules file, how to use the built-in rules, and how to code specialized rules for validating SWIFT documents.
- [Appendix B, “Handling Acknowledgements,”](#) describes the process of acknowledging a SWIFT document after it has passed through validation.
- [Appendix C, “Linking Business Applications to SWIFTAlliance,”](#) describes ways of connecting business applications to SWIFTAlliance.
- [Appendix D, “Adapter Support for AS1 and AS2 Communications,”](#) describes support for AS1 and AS2 communications, which are specifications for the independent transport of interchange documents over the Internet.

Who Should Read This Documentation

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

- **Integration Specialists**—Lead the integration design effort. Integration specialists have expertise in defining the business and technical requirements of integration projects, and in designing integration solutions that implement specific features of WebLogic Integration. The skills of integration specialists include business and technical analysis, architecture design, project management, and WebLogic Integration product knowledge.
- **Technical Analysts**—Provide expertise in an organization’s information technology infrastructure, including telecommunications, operating systems, applications, data repositories, future technologies, and IT organizations. The skills of technical analysts include technical analysis, application design, and information systems knowledge.
- **Enterprise Information System (EIS) Specialists**—Provide domain expertise in the systems that are being integrated using WebLogic Integration adapters. The skills of EIS specialists include technical analysis and application integration design.
- **System Administrators**—Provide in-depth technical and operational knowledge about databases and applications deployed in an organization. The skills of system administrators include capacity and load analysis, performance analysis and tuning, deployment topologies, and support planning.

Additional Information

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

- *BEA WebLogic Adapter for SWIFT Release Notes*
<http://edocs.bea.com/wladapters/swift/docs811/pdf/relnotes.pdf>
- *BEA WebLogic Adapter for SWIFT Installation and Configuration Guide*
<http://edocs.bea.com/wladapters/swift/docs811/pdf/install.pdf>
- *Introduction to the BEA WebLogic Adapters for WebLogic*
<http://edocs.bea.com/wladapters/docs81/index.html>
- *BEA WebLogic Adapters 8.1 Dev2Dev Product Documentation*
<http://dev2dev.bea.com/products/wladapters/index.jsp>

- Application Integration documentation
<http://edocs.bea.com/wli/docs81/aiover/index.html>
<http://edocs.bea.com/wli/docs81/aiuser/index.html>
- BEA WebLogic Integration documentation
<http://edocs.bea.com/wli/docs81/index.html>
- BEA WebLogic Platform documentation
<http://edocs.bea.com/platform/docs81/index.html>
- SWIFT documentation
<http://www.swift.com>

How to Use This Document

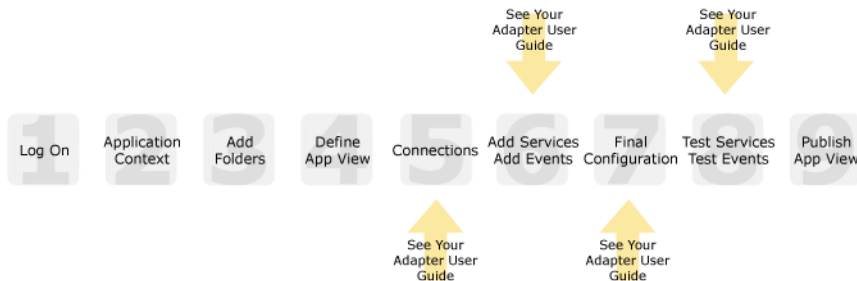
This document is designed to be used in conjunction with *Using the Application Integration Design Console*, available at the following URL:

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

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

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

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



Contact Us!

Your feedback on the BEA WebLogic Adapter for 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 BEA WebLogic Adapter for SWIFT documentation.

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

If you have any questions about this version of BEA WebLogic Adapter for SWIFT, or if you have problems using 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.
<i>italics</i>	Indicates emphasis or book titles.

Convention	Item
monospace text	<p>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.</p> <p><i>Examples:</i></p> <pre>#include <iostream.h> void main () the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre>
monospace boldface text	<p>Identifies significant words in code.</p> <p><i>Example:</i></p> <pre>void commit ()</pre>
<i>monospace italic text</i>	<p>Identifies variables in code.</p> <p><i>Example:</i></p> <pre>String <i>expr</i></pre>
UPPERCASE TEXT	<p>Indicates device names, environment variables, and logical operators.</p> <p><i>Examples:</i></p> <pre>LPT1 SIGNON OR</pre>
{ }	<p>Indicates a set of choices in a syntax line. The braces themselves should never be typed.</p>
[]	<p>Indicates optional items in a syntax line. The brackets themselves should never be typed.</p> <p><i>Example:</i></p> <pre>buildobjclient [-v] [-o name] [-f <i>file-list</i>]... [-l <i>file-list</i>]...</pre>
	<p>Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.</p>

Convention	Item
...	<p data-bbox="342 355 826 373">Indicates one of the following in a command line:</p> <ul data-bbox="342 390 1073 494" style="list-style-type: none"> <li data-bbox="342 390 1028 407">• That an argument can be repeated several times in a command line <li data-bbox="342 425 920 442">• That the statement omits additional optional arguments <li data-bbox="342 460 1055 477">• That you can enter additional parameters, values, or other information <p data-bbox="342 503 732 520">The ellipsis itself should never be typed.</p> <p data-bbox="342 546 436 564"><i>Example:</i></p> <pre data-bbox="342 581 1001 633">buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</pre>
.	<p data-bbox="342 668 1073 685">Indicates the omission of items from a code example or from a syntax line.</p> <p data-bbox="342 694 813 711">The vertical ellipsis itself should never be typed.</p>

Introducing the BEA WebLogic Adapter for SWIFT

This topic introduces the BEA WebLogic Adapter for SWIFT and describes how the adapter enables integration with SWIFT documents and WebLogic Integration. This section includes the following topics:

- [About the BEA WebLogic Adapter for SWIFT](#)
- [Getting Started With the Adapter for SWIFT](#)

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

About the BEA WebLogic Adapter for SWIFT

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 any of the BEA application and data adapters that are available from the adapter suite of products. The same adapters can be used to obtain information that is required to populate SWIFT documents, such as 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 document.

This section includes the following topics:

- [Supported Features for Application Integration](#)
- [Supported Services](#)

- [Supported Events](#)
- [Benefits of the Adapter for SWIFT](#)
- [Components of the Adapter Kit](#)
- [SWIFT Document Validation](#)

Supported Features for Application Integration

The BEA WebLogic Adapter for SWIFT provides:

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

The adapter provides pre-packaged support for standard SWIFT document 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.

Supported Services

The WebLogic Adapter for SWIFT supports the following types of services:

Table 1-1 Services Supported by the WebLogic Adapter for SWIFT

Service Type	Description
MQ service	Sends a SWIFT message to an IBM MQSeries or WebSphere MQ queue.
File service	Sends a SWIFT file to a specific directory on disk.
FTP service	Sends a SWIFT file via FTP.

To learn more about configuring services, see [“Setting Service Properties” on page 3-6](#).

Supported Events

The WebLogic Adapter for SWIFT supports the following types of events:

Table 1-2 Events Supported by the WebLogic Adapter for SWIFT

Service Type	Description
MQ Event	Adapter picks up a SWIFT message from a specific IBM MQSeries or WebSphere MQ queue
File Event	Adapter picks up a SWIFT file from a specific directory on disk.
FTP Event	Adapter picks up a SWIFT file via FTP.
TCP Event	Adapter picks up a SWIFT file via TCP.

To learn more about configuring events, see [“Setting Event Properties” on page 3-12](#).

Benefits of the Adapter for SWIFT

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

- Integration can be achieved without custom coding.
- Business processes can be started by events generated by SWIFT documents.
- Business processes can request and receive SWIFT documents using services.
- Adapter events and services are standards-based. The adapter services and events provide extensions to the *J2EE Connector Architecture* (JCA) version 1.0 from Sun Microsystems, Inc. To learn more about JCA, see the Sun JCA page at the following URL:

<http://java.sun.com/j2ee/connector/>

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

<http://edocs.bea.com/wls/docs81/cluster/index.html>

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

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

Components of the Adapter Kit

The WebLogic Adapter for SWIFT distribution (`BEA_SWIFT_8_1.ear`) contains preconfigured components that you can customize to support the integration of SWIFT document formats into your environment.

Table 1-3 Components of the WebLogic Adapter for SWIFT Kit

Component	Description
XML schemas (.xsd)	Describe the SWIFT documents for event and service processing.
Data dictionaries (.dic)	Describe the SWIFT format messages to the BEA WebLogic Adapter for SWIFT.
Transformation templates (.xch)	Maps that are created using the transformation templates configured to convert SWIFT to XML and XML to SWIFT.
Rules files (.xml)	XML documents that are used to apply business rules to the SWIFT XML file (either post or prior to conversion). The rules files conform to SWIFT specifications and are customizable to apply customer and/or partner specific rules. To learn more about rules, see Appendix A, “Validation Rules.”
Code sets (.txt)	Currency and country (ISO standard) code set tables are provided for validation using the rules engine.

To learn more about these components in the repository, see [“Contents of the Schema Repository” on page 2-4.](#)

Note: To customize the WebLogic Adapter for SWIFT in your environment, you must explicitly configure these components provided in the kit. For assistance, contact BEA Customer Support through BEA WebSUPPORT at www.bea.com.

SWIFT Document Validation

SWIFT documents are validated in two ways:

- Dictionaries are used to parse the document and validate the structure of the message type and tag structure.

SWIFT documents are described using data dictionaries supplied with the adapter. Data dictionaries describe the XML format and are used to enable the mapping of XML documents to SWIFT form and SWIFT documents to XML. The data dictionaries and the format for describing data elements conform to SWIFT standards. These dictionaries are in XML format and can be edited to tailor messages to individual bank and/or market standards.

- The rules engine 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.

BEA supplies standard rules files that validate a document based on the SWIFT 2001, 2002, and 2003 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 you use customized rules files and apply them to custom built transformation templates, refer to [Appendix A, “Validation Rules,”](#) which describes the standard rules and SWIFT-specific rules supplied with the BEA WebLogic Adapter for SWIFT.

Depending on the direction (XML-to-SWIFT or SWIFT-to-XML), the content validation occurs before (XML-to-SWIFT) or after (SWIFT-to-XML) structural validation.

Validation of Inbound SWIFT Documents

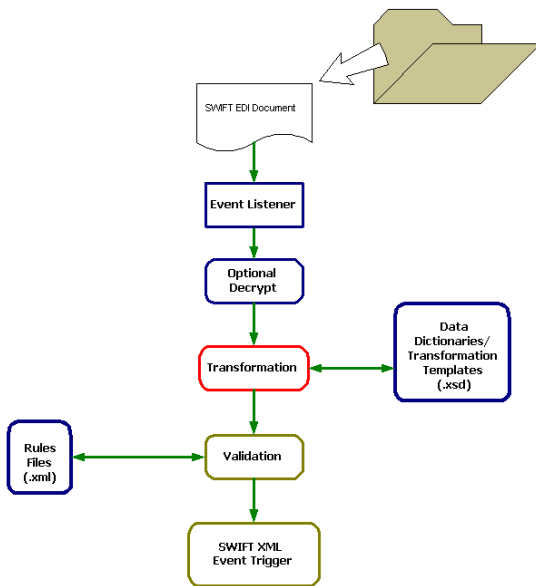
In the event of an inbound SWIFT document, a configured listener picks up the SWIFT document and then the following steps occur:

1. The adapter applies an encryption or decryption algorithm. This is a configurable process, and you can use any Java™-based encryption processor.
2. The adapter performs the pre-parse stage in which the document (if not in XML format at this stage) is converted to XML. At this stage, the pre-built (or customized) transformation templates are called to convert SWIFT format to XML.
3. After structural integrity has been verified during the transformation stage, the adapter performs content validation, using a set of rules defined in an XML formatted rules file. 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 SWIFT standards.

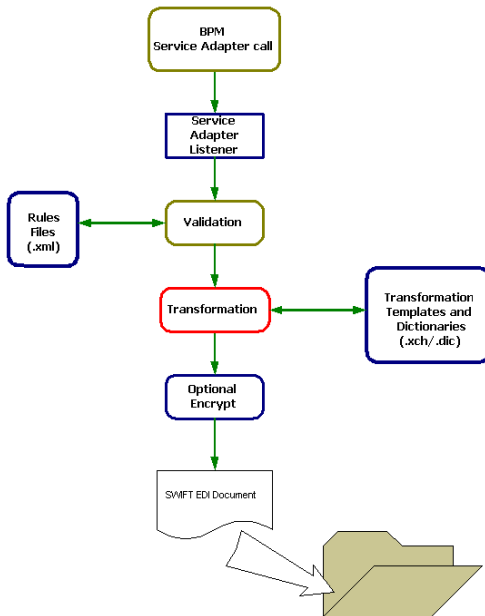
4. Where applicable, the adapter returns acknowledgment documents to the sending application, but only if the incoming document is structurally valid. If the content validation fails, an error code is returned in the acknowledgement document (when acknowledgement is expected).
5. After it is parsed and validated, the XML document is processed accordingly. The process flow can end here (for example, parse, convert, validate, and apply business logic).

Figure 1-1 Inbound Document Processing



Validation of Outbound SWIFT Documents

For outbound messages, 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

Post-Validation Acknowledgements

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 documents, the default Acknowledgement agent is the XDSWIFTACKAgent. The acknowledgement process is described in detail in [Appendix B, “Handling Acknowledgements.”](#)

Getting Started With the Adapter for SWIFT

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

- [Step 1: Design the Application Integration Solution](#)
- [Step 2: Determine the Required Business Processes for SWIFT Documents](#)
- [Step 3: Generate Schemas and Define Document Transformations](#)
- [Step 4: Define Application Views and Configure Services and Events](#)
- [Step 5: Define Validation for SWIFT Documents](#)
- [Step 6: Integrate with Other BEA Software Components](#)
- [Step 7: Deploy the Solution to the Production Environment](#)

Step 1: Design the Application Integration Solution

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

- Defining the overall scope of application integration.
- Determining the business process(es) to integrate.
- Determining which WebLogic Platform components will be involved in the integration, such as web services or business processes designed in WebLogic Workshop, portals created in WebLogic Portal, and so on.
- Determining which external systems and technologies will be involved in the integration, such as financial systems or other enterprise integration systems (EISs) that handle SWIFT messages.
- Determining which BEA WebLogic Adapters for WebLogic Integration will be required, such as the BEA WebLogic Adapter for SWIFT and adapters for any related EISs. An application integration solution can involve multiple adapters.

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

Step 2: Determine the Required Business Processes for SWIFT Documents

Within the larger context of an application integration project, you must determine which specific SWIFT documents and business processes are required to handle SWIFT documents for services and events to support the business processes in the application integration solution. Or, if you are invoking EIS-based business services or business components directly, rather than through a business process, you must determine the tasks you need to complete.

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

- Type of integration endpoints, business processes, and transport used to access the EIS that handles the SWIFT documents.
- Transactions involved in business processes
- Logins required to access and perform the required operations
- Whether operations are, from the adapter point of view:
 - services, which notify the external system, via SWIFT documents, with requests for action, and, in addition, whether such services should be processed synchronously or asynchronously
 - events, which are notifications from the external system, via SWIFT documents, that trigger business processes

This step involves the expertise of EIS and SWIFT specialists, including analysts and administrators.

Step 3: Generate Schemas and Define Document Transformations

After identifying the SWIFT integration objects and business processes required for the application integration solution, you must use utilities provided with the WebLogic Adapter for SWIFT to generate the XML schemas and define document transformations that will be used to manage SWIFT documents.

- Services require two generated XML schemas: one for the request and another for the response.
- Events require a single generated XML schema to handle SWIFT documents sent by the originating system.

To learn more about schemas and transformations, see [Chapter 2, “Transforming Document Formats.”](#)

Step 4: Define Application Views and Configure Services and Events

After you create the schemas for your SWIFT services or events, you create an application view that provides an XML-based interface between WebLogic Server and a particular EIS, such as financial systems, within your enterprise that handles the SWIFT documents. If you are accessing multiple EISs, you define a separate application view for each EIS that you want to access. To provide different levels of security access (such as “guest” and “administrator”), define a separate application view for each security level.

Once you define an application view, you can configure events and services in that application view that employ the XML schemas that you created in [“Step 3: Generate Schemas and Define Document Transformations” on page 1-9](#). To learn more about generating schemas and defining transformations, see [Chapter 2, “Transforming Document Formats.”](#)

To learn more about defining application views, see [Chapter 3, “Defining Application Views for SWIFT”](#) in conjunction with *Using the Application Integration Design Console*, at the following URL:

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

Step 5: Define Validation for SWIFT Documents

For inbound and outbound SWIFT documents, you can configure data dictionaries, specify validation rules, and define acknowledgements to notify external systems of the validation results. To learn more about SWIFT document validation, see [“SWIFT Document Validation” on page 1-5](#).

Step 6: Integrate with Other BEA Software Components

Once you have configured and published one or more application views for SWIFT integration, you can integrate these application views into other BEA software components, such as business processes or web services created in BEA WebLogic Workshop, or portals built with BEA WebLogic Portal.

To learn more about integration with other components, see *Using the Application Integration Design Console*, particularly Chapter 3, “Using Application Views with Business Processes,” at the following URL:

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

Step 7: Deploy the Solution to the Production Environment

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

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

Transforming Document Formats

This topic describes the process and mechanisms for transforming documents formats between SWIFT and XML, and between XML and XML. It contains the following topics:

- [About Schemas](#)
- [About Document Format Transformations](#)
- [About Schema Repositories](#)
- [Generating Transformation Templates and Document Schemas](#)
- [Next Steps](#)

About Schemas

Each service or event that the Adapter for SWIFT uses is defined by a schema. All of the documents the adapter sends to, or receives from, the EIS must be defined by schemas. The adapter uses the following schemas:

- [Service Requests](#)
- [Service Responses](#)
- [Events](#)

Service Requests

Service requests are requests for action that your application makes to your EIS. Requests are defined by request schema. As part of the definition, the request schema defines the input parameters required by the EIS. The EIS system responds to the request with a service response.

Service Responses

Service responses are the way the EIS responds to a service request. A service response schema defines this service response. Service requests always have corresponding responses.

Events

Events are generated by the EIS as a result of publishing SWIFT documents. You can use these events to trigger an action in your application. For example, the EIS may generate an event when customer information is updated. If your application must do something when this happens, your application is a consumer of this event. Events are defined by event schema.

About Document Format Transformations

Documents within WebLogic Integration are encoded in XML. However, you may need to receive and generate SWIFT documents. The WebLogic Adapter for SWIFT supports the following transformations:

- [Transforming SWIFT to XML \(Events Only\)](#)
- [Transforming XML to SWIFT \(Services Only\)](#)

You specify the type of transformation when adding an event or service to an application view, as described in [Chapter 3, “Defining Application Views for SWIFT.”](#)

Note: For XML-to-XML transformations (applicable to events and services), you need to use the Transformation Control in WebLogic Workshop. To learn more about the Transformation Control, see “Guide to Data Transformation” in the WebLogic Workshop online help system at:

<http://edocs.bea.com/workshop/docs81/doc/en/integration/dtguide/dtguideIntro.html>

Transforming SWIFT to XML (Events Only)

For events originating from SWIFT data sources, the WebLogic Adapter for SWIFT converts that data to XML for processing using business processes. This conversion includes “pre-parsing” the data into XML, which is then parsed itself for processing. SWIFT format messages arriving at the adapter are processed by the SWIFT pre-parser, which determines the SWIFT document type, and applies the correct SWIFT to XML format conversion.

Transforming XML to SWIFT (Services Only)

For services that interact with SWIFT data sources, the WebLogic Adapter for SWIFT converts XML documents (for example, from business processes) to SWIFT standard format. This conversion includes “pre-emitting” the data into the SWIFT, non-XML format, and then emitting the data to the SWIFT message handling system (MHS).

The BEA WebLogic Adapter for SWIFT knows the document type by the root element of its XML representation. 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).

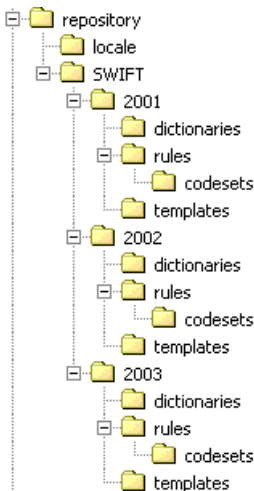
About Schema Repositories

This topic describes the schema repository, which stores schema information. It contains the following topics:

- [Contents of the Schema Repository](#)
- [About the Repository Manifest](#)
- [Naming Schema Repositories](#)
- [Modifying the Repository](#)

Contents of the Schema Repository

The `BEA_SWIFT_8_1.ear` file automatically generates repository directories and components.



A schema repository consists of the following elements:

- Manifest file (`manifest.xml`) that describes the event and service schemas contained in the repository. For an example, see `/SWIFT/2003/manifest.xml`.
- Event and service schemas. For examples, see `/SWIFT/2003/*.xsd`.

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 adapter. Schemas describe each request sent to and each response received from an 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.

- SWIFT document dictionaries. For examples, see `/SWIFT/2003/dictionaries/*.dic`.

SWIFT documents are described by dictionary files. For events, the adapter converts SWIFT documents into XML on entry and, for services, converts from XML to SWIFT on exit. Transformation from SWIFT to XML is described in [“Transforming SWIFT to XML \(Events Only\)” on page 2-3](#) and is based on a generalized transformation engine making use of dictionaries and `xch` transformation templates stored in `.xch` files.

- Transformation template for converting from and to SWIFT. For examples, see `/SWIFT/2003/templates/*.xch`. To learn more about transformation templates, see [“Generating Transformation Templates and Document Schemas”](#) on page 2-8.
- Content and network validation rules for the validation and acknowledgement process. For examples, see `/SWIFT/2003/rules/*.xml`. To learn more about validation rules, see [Appendix A, “Validation Rules.”](#)
- ISO code set for countries and currencies. For examples, see `/SWIFT/2003/rules/codeset/*.txt`.

When you use the WebLogic Integration Application View Console to create an Application View, a schema repository is automatically created for you. In addition, the Application View creation process also creates a repository manifest and extracts the schemas into the repository.

Note: To implement the WebLogic Adapter for SWIFT in your environment, you must explicitly configure these components provided in the kit. For assistance, contact BEA Customer Support through BEA WebSUPPORT at www.bea.com.

About the Repository Manifest

Each schema repository has a manifest that describes the repository and its contents. The repository manifest is an XML file named `manifest.xml`. This file is created automatically when the repository is generated from the `BEA_SWIFT_8_1.ear` file.

The following is an example of a manifest file showing the relationships between events and schemas and service request and response schemas.

Listing 2-1 Sample Repository Manifest (Portion Only)

```
</manifest>
  <schemaref name="MT540">
    <request root="SWIFTMT540" file="MT540.xsd"/>
    <response root="emitStatus" file="EmitStatus.xsd"/>
    <event root="SWIFTMT540" file="MT540.xsd"/>
  </schemaref>
</manifest>
```

The repository has a connection section, which can be ignored for this adapter. It also has a schema reference section, named `schemaref`. The schema reference name appears in the

drop-down list on the Add Service or Add Event screens in the WebLogic Integration Application View Console. Each named schema reference can contain three schemas, one of each type.

Naming Schema Repositories

The schema repository has a three-part naming convention.

- On UNIX:

session_base_directory/adapter/connection_name

- On Windows:

session_base_directory\adapter\connection_name

where

Table 2-1 Schema Repository Naming Convention

Name	Description
<i>session_base_directory</i>	Schema's session base path, which represents a folder under which multiple sessions of schemas can reside.
<i>adapter</i>	Type of adapter (for example, SWIFT).
<i>connection_name</i>	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 `2003`, then the schema repository is the directory:

`/usr/opt/bea/bse/SWIFT/2003`

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 generated from the supplied `BEA_SWIFT_8_1.ear` file into the directory you choose. Each time a new application view is created, the repository is generated from the `BEA_SWIFT_8_1.ear` file.

Modifying the Repository

During creation of the application view, the Session Path is supplied (see [Figure 2-1](#)), and any supplied connection repositories are extracted to the location you specify.

Figure 2-1 Configure Connection Parameters

The screenshot shows a configuration window with the following elements:

- Session Path*:** A text input field containing the path `D:\Adapters\swift\working`.
- Connection Name*:** A dropdown menu currently displaying `2003`.
- Connect to EIS:** A button located below the dropdown menu.

If a built-in manifest is found in the `BEA_SWIFT_8_1.ear` file, it is generated in the location you specify. Any duplicate files are overwritten. The Connection name drop-down list box reflects the contents of the directory after generation; 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 the `BEA_SWIFT_8_1.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 `BEA_SWIFT_8_1.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, `2003_Custom`).
2. Create a new application view supplying the same session path. At this point, the new connection (for example, `2003_Custom`) appears in the list and will not be overwritten by the original supplied connection repository.

Modify Repository and Update the EAR File

This procedure ensures that modifications become a permanent feature of the product by updating the supplied `BEA_SWIFT_8_1.ear` file.

To make changes to the repository:

1. Modify objects in the repository.
2. Load the `BEA_SWIFT_8_1.ear` file with the changed repository.
 - a. From the `BEA_SWIFT_8_1.ear` file, extract the `shared.jar` file.
 - b. From the extracted `shared.jar` file, extract the `BEA_SWIFT_8_1.manifest.zip`.
 - c. Replace/Update the files in `BEA_SWIFT_8_1.manifest.zip` with the modified objects (`manifest.xml`, dictionaries, transformation templates, rules, and code sets).
 - d. Replace/Update the `BEA_SWIFT_8_1.manifest.zip` in the `shared.jar`.
 - e. Replace the `shared.jar` in the `BEA_SWIFT_8_1.ear` file.

At this point, whenever a new application view is created, your modified connection repository will be generated out of the `BEA_SWIFT_8_1.ear` file (from the included `shared.jar`, from the `BEA_SWIFT_8_1.manifest.zip`).

Generating Transformation Templates and Document Schemas

This topic describes the sample utilities provided with the BEA WebLogic Adapter for SWIFT that generate transformation templates and document schemas. It includes the following sections:

- [About the Sample Utilities](#)
- [Extracting the Sample Utilities](#)
- [Generating Transformation Templates](#)
- [Generating Document Schemas](#)
- [Automatically Generating a Session Repository](#)

About the Sample Utilities

The BEA WebLogic Adapter for SWIFT provides the following sample utilities in the `BEA_SWIFT_SAMPLES.zip` file:

Table 2-2 Utilities for Generating Transformation Templates and Document Schemas

Utility	Description
<code>XCHZen.class</code>	Utility class (in the <code>LIB</code> subdirectory) that generates a transformation template from a SWIFT document described in a dictionary (<code>.dic</code> file).
<code>XSDZen.class</code>	Utility class (in the <code>LIB</code> subdirectory) that generates a W3C XML schema for the related XML document.
<code>genswift.cmd</code>	Command script that automatically generates all transformation templates and all W3C XML schemas for all dictionaries in a session repository.

Extracting the Sample Utilities

The sample utilities are packed with the associated `BEA_SWIFT_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_SAMPLES.zip samples
```

This command creates a `LIB` subdirectory at the current location.

Generating Transformation Templates

The `XCHZen.class` utility class generates transformation templates to transform from SWIFT to XML and from XML to SWIFT.

- To generate a SWIFT-to-XML template, use the following command:

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

- To generate an XML-to-SWIFT template, use the following command:

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

where *YourDictionary* is the name of the dictionary you want to use.

Generating Document Schemas

The `XSDZen.class` utility 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 described in the previous section, not on any custom mapping that a user may have performed. To generate document schemas, use the following command:

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

where *YourDictionary* is the name of the dictionary you want to use (the `.dic` extension is required).

Automatically Generating a Session Repository

The `genswift.cmd` utility automatically regenerates all transformation templates and document schemas for all sessions in a session path (see [“Extracting the Sample Utilities” on page 2-9](#) for instructions on extracting samples from the `BEA_SWIFT_SAMPLES.ZIP` file). From the session path, the `genswift.cmd` utility traverses all directories under `session_path/SWIFT` and, for each dictionary in the dictionary directory, generates an XML to SWIFT template, a SWIFT to XML template, and a document schema for the XML representation.

To generate a session repository automatically, use the following command:

```
samples_path/genswift.cmd
```

where *samples_path* is the location of the samples directory extracted in the section [“Extracting the Sample Utilities” on page 2-9](#). At this point, you may follow the instructions in [“Modify Repository and Update the EAR File” on page 2-8](#) to make the results permanent for your product.

Next Steps

After you have defined schemas for your events and services, the next step is to create an application view. An application view makes the services and events available to applications. To learn more about application views, see [Chapter 3, “Defining Application Views for SWIFT.”](#)

Defining Application Views for SWIFT

An application view is a business-oriented interface to objects and operations within an EIS. This section describes application views and contains the following topics:

- [How to Use This Document](#)
- [Before You Begin](#)
- [About Application Views](#)
- [About Defining Application Views](#)
- [Defining Service Connection Parameters](#)
- [Setting Service Properties](#)
- [Setting Event Properties](#)
- [Defining Event Connection Parameters](#)
- [Testing Services](#)
- [Testing Events Using a Service](#)
- [Testing Events Manually](#)

How to Use This Document

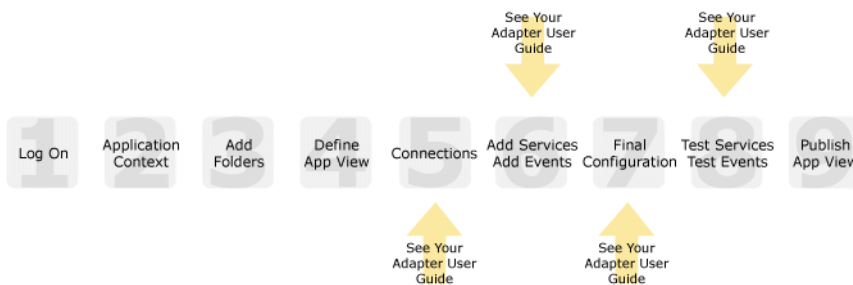
This document is designed to be used in conjunction with *Using the Application Integration Design Console*, available at the following URL:

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

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

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

Figure 3-1 Information Interlock with *Using the Application Integration Design Console*



Before You Begin

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

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

defining and testing services and events, see “[Getting Started With the Adapter for SWIFT](#)” on page 1-8.

- Gathered the connection information for your SWIFT system. To learn more about the connection information needed for your SWIFT system, see “[Modifying the Repository](#)” on page 2-7.

About Application Views

An application view defines:

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

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

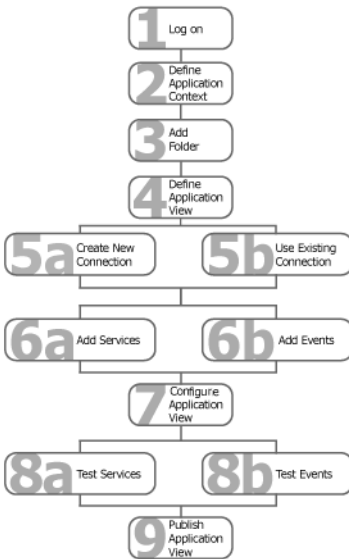
About Defining Application Views

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

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

The information you enter depends on the requirements of your business process and your EIS system configuration. [Figure 3-2](#) summarizes the procedure for defining and configuring an application view.

Figure 3-2 Process for Defining and Configuring an Application View



To define an application view:

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

This application will be using the events and services you define in your application view. The application view works within the context of this application.

3. Add folders as required to help you organize application views.
4. Define a new application view for your adapter.
5. Add a new connection service or select an existing one.

If you are adding a new connection, see [“Defining Service Connection Parameters” on page 3-5](#) for details about SWIFT requirements.

6. Add the events and services for this application view.

See the following sections for details about SWIFT requirements:

- [“Setting Service Properties” on page 3-6](#)

- “Setting Event Properties” on page 3-12
7. Perform final configuration tasks.

If you are adding an event connection, see “Defining Event Connection Parameters” on page 3-20 for details about SWIFT requirements.
 8. Test all services and events to make sure they can properly interact with the target EIS.

See the following sections for details about SWIFT requirements:

 - “Testing Services” on page 3-23
 - “Testing Events Using a Service” on page 3-26
 - “Testing Events Manually” on page 3-28
 9. Publish the application view to the target WebLogic Workshop application.

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

Defining Service Connection Parameters



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

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

The Select Browsing Connection page allows you to choose the type of connection factory to associate with the application view. You can select a connection factory within an existing instance of the adapter or create a connection factory within a new adapter instance. After you enter a connection name and description, you use the Configure Connection Parameters page to specify connection parameters for a connection factory.

To create a new browsing connection:

1. In the Create New Browsing Connections page, enter a connection name and description as described in *Using the Application Integration Design Console*.

The Configure Connection Parameters page appears to allow you to configure the newly created connection factory within the new adapter instance.

Session Path*

Connection Name*

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

2. Specify a session path and connection name.

This information enables the application view to interact with the target EIS. You need enter this information only once per application view.

3. Click Connect to EIS.

You return to the Create New Browsing Connections, where you can specify connection pool parameters and logging levels. To learn more about configuring browsing connections, see *Using the Application Integration Design Console* at the following URL:

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

Setting Service Properties

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

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

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

WebLogic Adapter for SWIFT uses services to make requests for SWIFT messages. Emission of SWIFT documents has been implemented as a service. A service consists of both a request and a response. The Adapter for SWIFT supports the following services:

- [MQ Service](#)
- [File Service](#)
- [FTP Service](#)

To learn more about these types of services, see “[Supported Services](#)” on page 1-2.

MQ Service

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

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

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

An MQ service sends a SWIFT document to an IBM MQSeries or WebSphere MQ queue.

To configure an MQ Service:

1. Enter a unique service name that describes the function the service performs.
2. Select MQEmit_SWIFT from the Select list.

The Add Services page displays the fields required for this service type.

Select:	MQEmit_SWIFT
Queue Manager*	<input type="text"/>
Queue Name*	<input type="text"/>
Correlation Id	<input type="text"/>
MQ Client Host	<input type="text"/>
MQ Client Port	<input type="text"/>
MQ Client Channel	<input type="text"/>
Polling Interval	<input type="text"/>

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

The properties correspond to the MQSeries communication settings that the adapter uses to communicate with MQSeries to write messages to the queues and with the transform options described in “[About Document Format Transformations](#)” on page 2-2.

3. Enter the following information:

Table 3-1 MQSeries Service Properties

Property	Description	Type
Queue Manager	Name of the MQSeries Queue Manager to be used.	string
Queue Name	Queue on which request documents are received.	string
Correlation Id	The correlation ID to set in the MQSeries message header.	string
MQ Client Host	For MQ Client only. Host on which MQ Server is located.	string
MQ Client Port	For MQ Client only. Port number to connect to an MQ Server.	integer
MQ Client Channel	For MQ Client only. Channel between an MQ Client and MQ Server.	string
Polling Interval	The time in milliseconds	duration

4. See “[Common Service and Event Settings](#)” on page 3-11 for information about selecting a schema and configuring tracing.

File Service



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

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

A File service sends a SWIFT document to a specific directory on disk.

To configure a File Service:

1. Enter a unique service name that describes the function the service performs.
2. Select `FileEmitter_SWIFT` from the Select list.

The Add Services page displays the fields required for this service type.

Select: FileEmitter_SWIFT

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

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

3. Enter the following information:

Table 3-2 File Service Properties

Property	Type / Value	Description
directory	Directory Path	Directory to which output messages are emitted.
output file name/mask	String	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.

4. See “[Common Service and Event Settings](#)” on page 3-11 for information about selecting a schema and configuring tracing.

FTP Service



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

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

An FTP service sends a SWIFT document through FTP to the target EIS.

To configure an FTP Service:

1. Enter a unique service name that describes the function the service performs.
2. Select FTPEmitter_SWIFT from the Select list.

The Add Services page displays the fields required for this service type.

Select: FTPEmitter_SWIFT ▼

Host name*	<input type="text"/>
Port number	<input type="text"/>
User Id*	<input type="text"/>
Password	<input type="text"/>
destination	<input type="text"/>
output file name/mask*	<input type="text"/>
Retry Interval	<input type="text"/>
Maxtries	<input type="text"/>

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

3. Enter the following information:

Table 3-3 FTP Service Settings

Setting	Type / Value	Description
Host name	String	FTP target system.
Port number	Numeric	FTP target system port (leave empty for FTP default).
User Id	String	User account ID to use when connecting to the FTP host.
Password	String	Password for the user account to use when connecting to the protocol host.
destination	String	Directory to address on the FTP target system.
output file name/mask	String	The output file name (can contain a '*'), which gets expanded to a timestamp.
Retry Interval	Retry Interval Duration	The maximum wait interval between retries when a connection fails. Retry interval duration in xxH : xxM : xxS format. For example: <ul style="list-style-type: none"> 1H:2M:3S is 1 hour 2 minutes and 3 seconds 0H:0M:30S is 30 seconds
Maxtries	Integer	Maximum number of retry attempts if a write failure occurs.

- See “[Common Service and Event Settings](#)” on page 3-11 for information about selecting a schema and configuring tracing.

Common Service and Event Settings

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

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

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

You select a schema and select tracing options the same way for all services.

To set common service settings:

- In the Schema list, select the schema you want to use with this service.

To learn more about document format transformations, see [Chapter 2, “Transforming Document Formats.”](#)

schema:

- Configure tracing for this service.

Tracing displays runtime information in the console. You set the type and amount of information you wish to capture as part of the final configuration tasks. This is described in detail in *Using the Application Integration Design Console*.

settings	
Trace on/off	<input type="checkbox"/>
Verbose Trace on/off	<input type="checkbox"/>
Document Trace on/off	<input type="checkbox"/>
root to transform template directory	D:/Adapters/swift/working/SWIFT/20
root to XML Style sheet directory	D:/Adapters/swift/working/SWIFT/20

Table 3-4 Log Settings for Services and Events

Setting	Description
Trace	Select to enable tracing for this service, or clear to disable tracing.
Verbose Trace	Select to enable verbose tracing for this service, or clear to disable tracing.

Table 3-4 Log Settings for Services and Events (Continued)

Setting	Description
Document Trace	Select to enable document tracing for this service, or clear to disable tracing.
root to transform template directory	Root directory for document transformation templates (*.xch).
root to XML Style sheet directory	Root directory for XML style sheets.

3. Click Add to add the service.

To learn more about the next step, see *Using the Application Integration Design Console* at the following URL:

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

Setting Event Properties



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

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

An event defines how your application responds to incoming SWIFT documents. The WebLogic Adapter for SWIFT handles the receipt of SWIFT documents. The WebLogic Adapter for SWIFT supports the following events:

- [MQ Event](#)
- [File Event](#)
- [FTP Event](#)
- [TCP Event](#)

To learn more about these types of events, see “Supported Events” on page 1-3.

MQ Event

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

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

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

In an MQ Event, the adapter picks up a SWIFT document from a specific IBM MQSeries or WebSphere MQ queue and passes it to an event variable that is set in a business process.

To configure an MQ Event:

1. Enter a unique event name that describes the function the event performs.
2. Select `MQ_SWIFT` from the Select list.

The Add Events page displays the fields required for this event type.

Select: <code>MQ_SWIFT</code> ▼	
Queue Manager*	swiftQM
Queue Name*	event_q1
MQ Client Host	mqServer
MQ Client Port	1118
MQ Client Channel	svr_conn_chn
Polling Interval	1
ackagent	XDSWIFTACKAgent
Character Set Encoding*	UTF-8

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

3. Enter the following information:

Table 3-5 MQSeries Event Properties

Property	Description	Type	Sample Value
Queue Manager	Name of the MQSeries Queue Manager to be used.	string	OMBEA
Queue Name	Queue where request documents are received.	string	TEST.iO
MQ Client Host	For MQ Client only. Host on which MQ Server is located.	string	
MQ Client Port	For MQ Client only. Port number to connect to an MQ Server.	integer	
MQ Client Channel	For MQ Client only. Channel between an MQ Client and MQ Server.	string	
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	
ackagent	The agent responsible for processing acknowledgement or non-acknowledgment of SWIFT documents. The default acknowledgment agent generates an empty acknowledgment document or lists all failed validation rules in the non-acknowledgment document.	string	
Character Set Encoding	Sets the character set encoding to be used.	string	

4. See [“Common Service and Event Settings” on page 3-11](#) for information about selecting a schema and configuring tracing.

File Event

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

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

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

In a File Event, the adapter picks up a SWIFT document from a specific location on disk and passes it to an event variable that is set in a business process.

To configure a File Event:

1. Enter a unique event name that describes the function the event performs.
2. Select `File_SWIFT` from the Select list.

The Add Events page displays the fields required for this event type.

Select: `File_SWIFT` ▼

Location*	D:\Adapters\swift\working\in_out\in
File Suffix*	SWIFT
Polling Interval	1
Sort*	<input checked="" type="checkbox"/>
Scan sub-directories	<input type="checkbox"/>
File-read limit (per scan)	
Copy to Directory	
ackagent	XDSWIFTACKAgent
Character Set Encoding*	UTF-8

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

3. Enter the following information:

Table 3-6 File Event Properties

Setting	Type / Value	Description
Location	String	Location and name of the file pattern for the SWIFT document.
File Suffix	String	Mandatory suffix of the input file.
Polling Interval	Duration	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
Sort	Boolean	Flag to indicate whether the list of files pulled from the location should be sorted for processing order.
Scan sub-directories	Boolean	Flag to indicate whether files should be read from subdirectories.
File-read limit (per scan)	Integer	The number of files read per sweep of the File directory location.
Copy to Directory	String	Directory to which the document being processed will be copied. This allows you to keep copies of the documents.
ackagent	String	The agent responsible for processing acknowledgement or non-acknowledgment of SWIFT documents. The default acknowledgment agent generates an empty acknowledgment document or lists all failed validation rules in the non-acknowledgment document.
Character set encoding	String	Document character set.

4. See [“Common Service and Event Settings” on page 3-11](#) for information about selecting a schema and configuring tracing.

FTP Event

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

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

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

In an FTP Event, the adapter picks up a SWIFT document via FTP and passes it to an event variable that is set in a business process.

To configure an FTP Event:

1. Enter a unique event name that describes the function the event performs.
2. Select FTP_SWIFT from the Select list.

The Add Events page displays the fields required for this event type.

Select:

Host name*	172.16.192.75
Host port	
User Id*	lykalidas
Password*	*****
Location*	/swift/inFtp
File Suffix	swift
Polling interval	1
ackagent	XDSWIFTACKAgent
Character Set Encoding*	UTF-8

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

3. Enter the following information:

Table 3-7 FTP Event Properties

Setting	Type / Value	Description
Host name	String	FTP target system.
Host port	Integer	Alternate port number.
User ID	String	User account ID to use when connecting to protocol host.
Password	String	Password for user account to use when connecting to protocol host.
Location	String	Location on FTP server for source files.
File Suffix	String	Suffix of source files.
Polling Interval	Duration	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.
ackagent	String	The agent responsible for processing acknowledgement or non-acknowledgment of SWIFT documents. The default acknowledgment agent generates an empty acknowledgment document or lists all failed validation rules in the non-acknowledgment document.
Character Set Encoding	String	Sets the character set encoding to be used.

4. See [“Common Service and Event Settings” on page 3-11](#) for information about selecting a schema and configuring tracing.

TCP Event

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

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

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

In a TCP Event, the adapter picks up a SWIFT document via TCP and passes it to an event variable that is set in a business process.

To configure a TCP Event:

1. Enter a unique event name that describes the function the event performs.
2. Select TCP_SWIFT from the Select list.

The Add Events page displays the fields required for this event type.

Select:	TCP_SWIFT ▼
TCP/IP Port*	<input type="text"/>
Allowable client host	<input type="text"/>
ackagent	<input type="text"/>
Character Set Encoding*	<input type="text"/>

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

3. Enter the following information:

Table 3-8 TCP Event Properties

Setting	Type / Value	Description
TCP/IP Port	Integer	TCP listening port.
Allowable Client Host	String	Host name or host address of client restricted to accessing events for this adapter.

Table 3-8 TCP Event Properties (Continued)

ackagent	String	The agent responsible for processing acknowledgement or non-acknowledgment of SWIFT documents. The default acknowledgment agent generates an empty acknowledgment document or lists all failed validation rules in the non-acknowledgment document.
Character set encoding	String	Document character set.

4. See “[Common Service and Event Settings](#)” on page 3-11 for information about selecting a schema and configuring tracing.

Defining Event Connection Parameters



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

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

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

To define event connection parameters:

1. In Connections area on the Application View Administration page, click Select/Edit.
2. In the Event Connection area, click Select Existing to get a list of available connections.

Adapter Instances:

- **f.e_Default**

- Event Connections:

Connection	Selected	Description
Event	<input checked="" type="checkbox"/>	

- **f.r_Default**

- Event Connections:

Connection	Selected	Description
Event	<input checked="" type="checkbox"/>	

- **f.s_Default**

- Event Connections:

Connection	Selected	Description
Event	<input checked="" type="checkbox"/>	

3. Click the Event connection and click OK.

The Edit Event Adapter page allows you to define event parameters and configure the information that will be logged for the connection factory.

Connection Name: * a_Default_Event
 Description: 

Connection Parameters:

*

Log Configuration

Set the log verbosity level for this ConnectionFactory.

*

Select one of the following settings for the log:

- Log errors and audit messages
- Log warnings, errors, and audit messages
- Log informational, warning, error, and audit messages
- Log all messages

Note: For maximum tracing, select Log all Messages. This is the recommended setting to use when you are collecting debugging information for BEA support.

The log file (BEA_SWIFT_8_1_1.log) is stored in the directory from which the server was started.

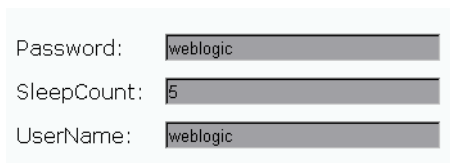
The following table describes the type of information that each logging message contains.

Table 3-9 Logging message categories

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

4. On the Edit Event Adapter page, click the Define button.

The Configure Event Delivery Parameters page appears.



The screenshot shows a form with three input fields. The first field is labeled 'Password:' and contains the text 'weblogic'. The second field is labeled 'SleepCount:' and contains the number '5'. The third field is labeled 'UserName:' and contains the text 'weblogic'.

5. Enter the following information:

Table 3-10 Event Connection Parameters

Parameter	Description
Password	Password for your WebLogic Server Administration Console user name.
SleepCount	Number of seconds the adapter will wait between polling for events.
Username	Your WebLogic Server Administration Console user name, defined in the <code>startWebLogic</code> script.

The event delivery parameters you enter on this page enable connection to your EIS and are used when generating events. The parameters are specific to the associated adapter and are defined in the `wli-ra.xml` file within the base adapter.

- Click Save to save your event delivery parameter settings. Click Continue to return to the Edit Event Adapter page, and then click Back to return to the Final Configuration and Testing page.

Testing Services

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

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

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

The purpose of testing an application view service is to evaluate whether that service interacts properly with the target EIS. When you test a service, you supply any inputs required to start the service. For the Adapter for SWIFT, the input is in the form of a valid XML string that acts as input for the service.

As an example, for `MQEmit_Service`, the 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.

To test SWIFT services:

- Confirm that the queue is empty where the 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.

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.excel.oQ	0	Local	
TEST.iQ	0	Local	
TEST.oQ	1	Local	
TEST.pQ	0	Local	
eQ	0	Local	
iQ	0	Local	
oQ	0	Local	

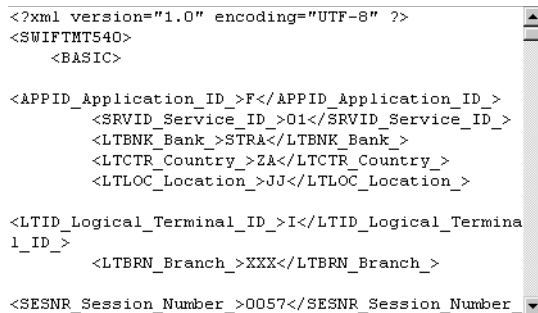
2. In the Application View Administration page, click the Test link beside the service to be tested.

The Test Services page appears.

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:

```
<?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. In the Test Service window, copy the appropriate XML strings from the SWIFT document for your account.



```
<?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_>
```

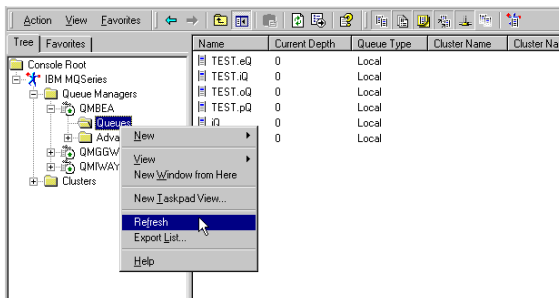
5. Click Test.

The results appear in the Test Results window.

The response document should indicate the success of emitting to MQSeries, as shown in the following example:

```
<?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>
```

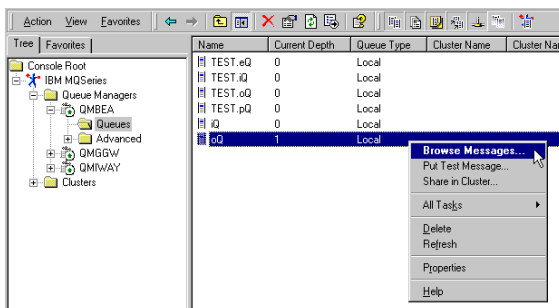
After transformation and formatting, the document should have arrived on the MQSeries queue. If you are testing an MQ service, you can also check your MQSeries Explorer for results (select Queue Managers>MyQueueManager>Queues). If the service ran correctly, the queue now contains the SWIFT message.



6. Check for the output document on the queue designated in configuring the service.
 - a. Using MQSeries Explorer, browse to the appropriate queue.
 - b. Select Refresh:

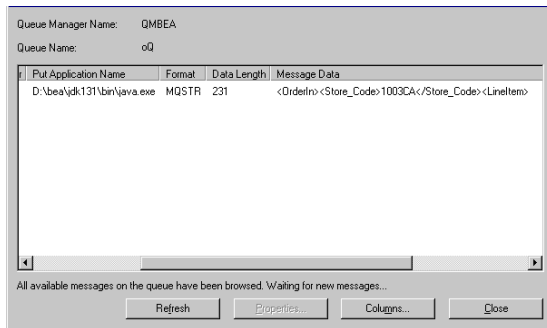
The Current Depth should be incremented.

There should be one if you cleared all messages first.



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 SWIFT adapter.



Testing Events Using a Service

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

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

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

The purpose of testing an application view event is to make sure that the adapter correctly handles events generated by SWIFT. When you test an event, you can trigger the event using a service or manually.

To test an event:

1. In the Application View Administration page, click the Test link below the Services row. The Test Events page appears.

Name: a
 Description:
 Status: Testing
 Published?: Not Published
 Available Actions: [Stop Testing](#) [Publish](#) ?

Connection Events and Services Variables

Events

fileE	Test View Summary View Event Schema
-------	---

Services

fileEs	Test View Summary View Request Schema View Response Schema
--------	---

2. Click Test for the required Event to display a screen similar to the following:

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?

Service

Manual

How long should we wait to receive the event?

Time (in milliseconds):

3. Select Service and select a service that triggers the event you are testing.
4. In the Time field, enter a reasonable period of time to wait, specified in milliseconds, before the test times out (One second = 1000 milliseconds. One minute = 60,000 milliseconds.).
5. Click Test and enter the XML string needed to trigger the service.

The service is executed.

- If the test succeeds, the Test Result page appears, showing the event document and the service output document.

Output from service fileS on application view f.s

```
<?xml version="1.0"?>
<emitStatus>
  <protocol>FILE</protocol>

  <parms>directory=D:\Adapters\swift\working\in_out\
out, pattern=kali#.swift</parms>
  <status>0</status>
  <msg/>
  <timestamp>2003-10-17T13:57:16:890Z</timestamp>
  <attempts>1</attempts>

  <name>D:\Adapters\swift\working\in_out\out\kali2.s
swift</name>
</emitStatus>
```

- If the test fails, the Test Result page displays only a Timed Out message.

Testing Events Manually



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

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

To test an event manually:

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?

- Service
- Manual

How long should we wait to receive the event?

Time (in milliseconds):

1. Select Manual.
2. In the Time field, enter a reasonable period of time to wait, specified in milliseconds, before the test times out (One second = 1000 milliseconds. One minute = 60,000 milliseconds.).
3. Click Test. The test waits for an event to trigger it.

4. Submit a valid SWIFT file into the expected File System directory, FTP location, or MQ Series Queue.
5. Either of the following should occur:
 - If the test succeeds, the Test Result page appears. This page displays the event document from the application, the service input document, and the service output document.

Generated event of type fileE on application view ap

```
<?xml version="1.0"?>
<SWIFTMTS40>
  <BASIC>

  <APPID_Application_ID_>F</APPID_Application_ID_>
  <SRVID_Service_ID_>24</SRVID_Service_ID_>
  <LTBNK_Bank_>VNDZ</LTBNK_Bank_>
  <LTCTR_Country_>BE</LTCTR_Country_>
  <LTLOC_Location_>T2</LTLOC_Location_>

  <LTID_Logical_Terminal_ID_>B</LTID_Logical_Terminal_ID_>
  <LTBRN_Branch_>XXX</LTBRN_Branch_>

  <SESNR_Session_Number_>0001</SESNR_Session_Number_>
```

Input to service fileEs on application view ap

```
<?xml version="1.0"?>
<SWIFTMTS40>
  <BASIC>

  <APPID_Application_ID_>F</APPID_Application_ID_>
  <SRVID_Service_ID_>24</SRVID_Service_ID_>
  <LTBNK_Bank_>VNDZ</LTBNK_Bank_>

  <LTCTR_Country_>BE</LTCTR_Country_>

  <LTLOC_Location_>T2</LTLOC_Location_>

  <LTID_Logical_Terminal_ID_>B</LTID_Logical_Terminal_ID_>
```

Output from service fileEs on application view ap

```
<?xml version="1.0"?>
<emitStatus>
  <protocol>FILE</protocol>

  <parms>directory=D:\Adapters\swift\working\in_out\
in, pattern=kali#.swift</parms>
  <status>0</status>
  <msg/>
  <timestamp>2003-10-17T14:14:00:109Z</timestamp>
  <attempts>1</attempts>

  <name>D:\Adapters\swift\working\in_out\in\kali1.swift</name>
</emitStatus>
```

- If the test fails or takes too long, the Test Result page appears, showing a Timed Out message.

Validation Rules

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 needed to validate SWIFT documents. The following topics describe how to encode a rules file, how to use the built-in rules, and how to code specialized rules:

- [About the Rules File](#)
- [Writing Rules in Java](#)
- [Writing Rule Search Routines in Java](#)
- [General Validation Rules Reference](#)
- [SWIFT Specific Rules Reference](#)

About the 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>` tag and under this tag are `<rule>` tags, which may be enclosed within `<using>` tags. For example, for SWIFTMT540, a portion of the rules file might look like this:

Listing A-1 Portion of Sample Rules File for SWIFTMT540

```
<!-- SWIFT Standards Version : SWIFT 2003 -->
<SWIFTMT540>
  <using class="com.ibi.swift.XDSWIFTRules" radix=",">
    <!-- Schema Validation for Domain,Tag Syntax and Structural
Validations -->
    <rule name="checkSchema" tag="SWIFTMT540" full="no"
method="checkSchema" schema="/SWIFT/schemas/MT540.xsd" />
    <!-- Network Validations -->
    <rule name="R190" tag="SWIFTMT540._540.A.A1._22F"
method="checkValue" tagset="QUALF_Qualifier_, INDCR_Indicator_"
code="1=LINK/2=AFTE;BEFO;INFO;WITH" errorcode="T20" />

    ...

  </using>
</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.

Note: For assistance with customizing rules files, contact BEA Customer Support through BEA WebSUPPORT at www.bea.com.

<document> tag

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

Table A-1 <document> tag

Attribute	Use
<code>ackagent</code>	<p>Java program class called to construct the acknowledgement. For the BEA WebLogic Adapter for SWIFT, this class is the <code>XDSWIFTACKAgent.class</code>. The actual agent to be used is selected in a search order:</p> <ol style="list-style-type: none"> 1. The document specification in the repository. 2. This attribute of the outer tag of the rule file. 3. The listener configuration. <p>As soon as an ackagent is located, it is selected for use, and the search ends.</p>

<using> Tag

Table A-2 <using> tag

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

<rule> tag

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 include:

Table A-3 <rule> Tag

Attribute	Description
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, an stag must be used.
stag	This is a specification subsection tag.
name	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	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	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, as shown in the following example:

Listing A-2 Section of a Dictionary That Defines Precedence for Rules Files

```

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

```

```

SWIFTMT100
<in_validation>
  100RULES
</in_validation>
</document>

```

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:

Table A-4 Parameters Passed in Rules

Parameter	Description
Node	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	Data value of the addressed node. This differs from the <code>node.getValue()</code> return if the tag contained a subfield address (for example, <code>tag=x:2</code>).
Attributes	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-3 Node Checking Example

```

import java.util.*;
import com.ibi.edaqm.*;
public class XDMYRules extends XDRuleClass
{

```

Validation Rules

```
public XDMRules()
{
}
public void specialRule(XDNode node, String value,
                      HashMap attributes)
    throws XDError
{
    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 XDError(XD.RULE, XD.RULE_VIOLATION, "node value
"+value+" is not 'Value'");
    }
}
}
```

Rule violations should throw an `XDError` describing the violation.

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

Table A-5 Services for Preparing Rules

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

Table A-5 Services for Preparing Rules (Continued)

Method	Purpose
Boolean is InList (string list, string value).	Value must be in the list.
Void trace (int level, string msg).	Text of the message is written to the system trace file. The level should be one of the following values: <ul style="list-style-type: none"> • XD.TRACE_DEBUG • XD.TRACE_ERROR • 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 XDEceptions 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 following elements:

Table A-6 Elements in the EDIError Object

Element	Description
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	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 shutdown. 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 the following 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-4 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
    {

```

Validation Rules

```
        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);
    }
}

```

General Validation Rules Reference

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 following rules are general rules:

- [isN](#)
- [isR](#)
- [isDate](#)
- [isTime](#)

isN

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

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

Table A-7 isN Attributes

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

isR

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

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

Table A-8 isR Attributes

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	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

The `isDate` rule validates that a node is in CCYYMMDD format.

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

Table A-9 isDate Attributes

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

isTime

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

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

Table A-10 isTime Attributes

Attribute	Meaning
None	None.

SWIFT Specific Rules Reference

The following rules are specific to SWIFT:

- [isValidReference](#)
- [isValidISIN](#)
- [isNotPresent](#)
- [isValidMultiLine](#)
- [isSWIFTReal](#)
- [isSWIFTDate](#)
- [IsValidSWIFTString](#)
- [isSWIFTTime](#)
- [isValidMessageType](#)
- [checkValue](#)
- [checkCD](#)
- [checkRepetitive](#)
- [checkNodes](#)
- [checkChildSequence](#)
- [checkAddition](#)
- [checkRelation](#)
- [checkSegment](#)

isValidReference

The `isValidReference` rule 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

The `isValidISIN` rule 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

The `isNotPresent` rule validates whether or not the value is present.

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

isValidMultiLine

The `isValidMultiLine` rule 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 A-11 isValidMultiLine Attributes

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	Error code for the rule.

isSWIFTReal

The `isSWIFTReal` rule 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 A-12 isSWIFTReal Attributes

Attribute	Description
Min	Minimum number of characters.
Max	Maximum number of characters.
Errorcode	Error code for the rule.

isSWIFTDate

The `isSWIFTDate` rule 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 A-13 isSWIFTDate Attributes

Attribute	Meaning
Format	Format of the date to check against.
Errorcode	Error code for the rule.

IsValidSWIFTString

The `IsValidSWIFTString` rule validates the value according to the format specified.

The following table describes the valid values of the format:

Table A-14 Valid Values for IsValidSWIFTString

Value	Description
x	S.W.I.F.T X character list.
y	S.W.I.F.T Y character list.
z	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

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

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

SWIFT Z Character Set

A to Z (uppercase)

a to z (lowercase)

0 to 9

SWIFT Z Character Set

/ (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 A-15 isValidSWIFTString Attributes

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

isSWIFTTime

The `isSWIFTTime` rule 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 A-16 isSWIFTTime Attributes

Attribute	Meaning
Format	Type of date format to check against.
Errorcode	Error code for the rule.

isValidMessageType

The `isValidMessageType` rule 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 A-17 isValidMessageType Attribute

Attribute	Meaning
Errorcode	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

The `checkValue` rule 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 A-18 checkValue Attributes

Attribute	Meaning
Tagset	List of nodes to check against.
Code	Expression used to validate the elements.
Errorcode	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

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

Table A-19 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	First element must be present if there are no repetitive sequences of the second or third element.
F	Repetitive Sequence Related	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.
M	Mandatory	One of the children of the specified elements must contain a value.

Table A-20 checkCD Attributes

Attribute	Meaning
Tagset	List of nodes to check against.
Cd	Expression used to validate the elements.
Errorcode	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

The `checkRepetitive` rule 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 A-21 checkRepetitive Attributes

Attribute	Meaning
Search	Node to search for and check.
Errorcode	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

The `checkNodes` rule 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 A-22 checkNodes Attributes

Attribute	Meaning
Tagset	Two sets of nodes to check separated by ";".
Errorcode	Error code for the rule.

checkChildSequence

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

Table A-23 checkChildSequence Attributes

Attribute	Meaning
Start	Node to check if it is repetitive.

Table A-23 checkChildSequence Attributes (Continued)

Attribute	Meaning
Pattern	Expression used to validate the elements.
Errorcode	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

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

Table A-24 checkAddition

Attribute	Meaning
Tagset	List of nodes to check.
Errorcode	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 document. 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

The `checkRelation` rule 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 A-25 checkRelation Attributes

Attribute	Meaning
Tagset	Node to check.
Taglist	List of nodes to check.
Find	Node to find to see if it should be present.
Errorcode	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 A-26 checkSegment Attributes

Attribute	Meaning
Parent	Parent node.
Subseq	Sub-segment node.
Child	Child node.
Check	Node to check if present.
Errorcode	Error code for the rule.

Handling Acknowledgements

This section describes the process of acknowledging a document after it has passed through validation and includes the following topics:

- [About Acknowledgement Processing](#)
- [Processing Documents With Validation and Acknowledgement](#)
- [About the Acknowledgement Agent](#)
- [Acknowledgement Message 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 might generate errors and might not pass specific validation rules. The validation engine and document validation rules are described in full in [Appendix A, “Validation Rules.”](#)

About Acknowledgement Processing

The acknowledgement process indicates the receipt and validity of the received document. The WebLogic Adapter for SWIFT provides the following features that support the acknowledgement process:

- Validation

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

- 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 element level information. Validation errors are stored in the document tree and are available to the acknowledgement agent.

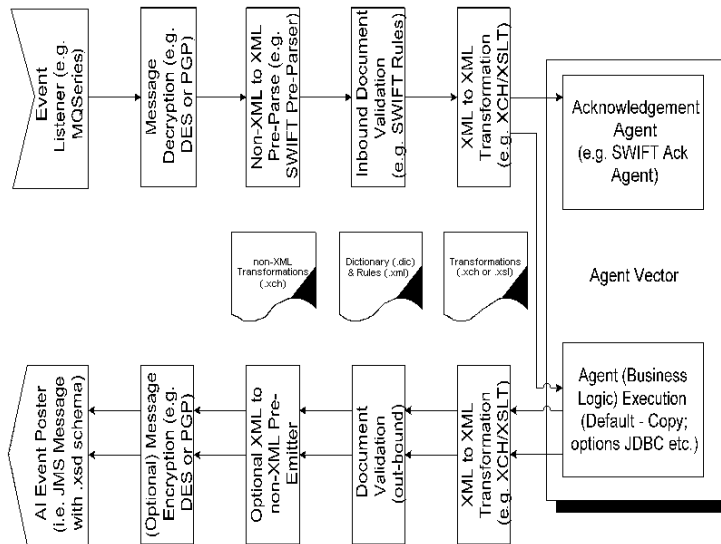
- Acknowledgement Agent

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, one, or more validation errors. The agent determines what constitutes a good document, to which an acknowledgement (ACK) should be sent. The agent also determines what constitutes a bad document, to which a non-acknowledgement (NAK) should be sent. The agent determines the content of the acknowledgement or non-acknowledgement.

Processing Documents With Validation and Acknowledgement

Documents proceed from the adapter event listener to the emitter or event poster and are processed in stages, as shown in the following figure:

Figure B-1 Document Life Cycle



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 includes both structural validation (described in dictionaries) and content and network validation (described in rules XML files).
- The validation processor (class) is defined at the document level or at the rule level, as described in [Appendix A, “Validation Rules.”](#) An example of a validation processor is `XDSWIFTRules.class`.
- Validation (particular dictionary and rules XML file) is tied to a specific XML document type.
- 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. For a discussion of acknowledgement agent determination, see [“About the Acknowledgement Agent” on page B-3.](#)
- Validation processing adds error elements to 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 (they have different schema, separate threads of execution, and so on).
- Validation errors and document output are independent of one another. A document may fail validation rules and have acknowledgements or non-acknowledgements generated in the acknowledgement agent. However, the document is still passed to its agent and sent to its output unless other specific actions are taken (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 can be coded to alter behavior when validation errors are present in the document tree.

About the Acknowledgement Agent

The acknowledgement agent is defined by an `<ackagent>` entry. It can be defined at the event or service level, or as an attribute in the root element of the rules XML file. The search order is as follows:

1. An attribute of the root element of the rules XML file.
2. The event or service listener definition using the Application View Console.

For the BEA WebLogic Adapter for SWIFT, the default ackagent is the `XDSWIFTACKAgent`. This setting is defined when you configure the service or event in the Application Integration Design Console, as described in [“Setting Service Properties” on page 3-6](#) and [“Setting Event Properties” on page 3-12](#), respectively.

Acknowledgement Message Handling

The validation engine performs the structural, content, and network validation rules defined in the document specific dictionaries and rules files. The acknowledgement agent generates an acknowledgement message based on the document tree, which is a composite of the original XML document tree and 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 listing shows sample output (for file or FTP services, output delivered into the directory; for MQ services, into the queue) with errors.

Listing B-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 listing shows a sample schema for handling the results of this acknowledgement.

Listing B-2 Sample Schema for Handling Schema Results

```
<?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, two messages traversing the system 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.

When defining the event in the application view (as described in [“Setting Event Properties” on page 3-12](#)), you need to set the ackagent value to the desired acknowledgement agent. In addition, an ACKNAK schema in the Schema drop down list box handles both acknowledgements and non-acknowledgements. For events, it is important that the adapter’s protocol settings are identical to those provided for the original event. If the settings are different, a separate Event Listener is created, the two events (document and associated ACK/NAK) are not tied together, and the ACKs/NAKs created by the document will not be seen by the event or schema combination created in this section.

After you create a SWIFT adapter with two registered events—a SWIFT document 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 WebLogic Workshop.

Handling Acknowledgements

Linking Business Applications to SWIFTAlliance

This section describes ways of connecting business applications to SWIFTAlliance. It includes the following topics:

- [Connecting Business Applications to SWIFTAlliance](#)
- [Connectivity Options](#)

To learn more about SWIFTAlliance, go to the following URL: <http://www.swift.com>.

Connecting Business Applications to SWIFTAlliance

SWIFT allows you to connect business applications to SWIFTAlliance in the following ways:

Table C-1 Ways to Connect Business Applications

Approach	Description
Manual File Transfer	S.W.I.F.T. messages are exchanged between the business application and SWIFTAlliance in batch files and with operator intervention on (either and/or both) the business application and the SWIFTAlliance.

Table C-1 Ways to Connect Business Applications (Continued)

Automated File Transfer	S.W.I.F.T. messages are exchanged between the business application and SWIFTAlliance in batch files, <i>without operator intervention</i> — 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 (via a conversational protocol) between the business application and SWIFTAlliance on an individual basis and <i>without operator intervention</i> on both SWIFTAlliance and Business Application.

Connectivity Options

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.

Table C-2 Connectivity Options In Conjunction With Other Adapters

Mode	BEA Adapter	Third Party Software
File Transfer	<ul style="list-style-type: none"> BEA WebLogic Adapter for SWIFT BEA WebLogic Adapter for File 	None specific but could be application, for example, MERVA or BESS
Application Server	<ul style="list-style-type: none"> BEA WebLogic Adapter for SWIFT BEA TCP/IP Adapter 	'AI MXA TCP-IP' CASmf Application Server
Interactive	<ul style="list-style-type: none"> 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, —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 via 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 to 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 customize.

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. acknowledgement and non-acknowledgements, and recording of events in the SWIFTAlliance journal. It can handle 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.

Adapter Support for AS1 and AS2 Communications

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). This topic describes BEA WebLogic Adapter for SWIFT support for AS1 and AS2 communications. It contains the following sections:

- [About the AS1 and AS2 Standards](#)
- [Comparison of AS1 \(SMTP/e-mail\) and AS2 \(HTTPS\)](#)
- [Adapter Support for AS1 and AS2](#)

Note: Support for AS1 and AS2 standards is not automatically provided with the WebLogic Adapter for SWIFT. In order to implement AS1 and AS2 in your environment, you must contact BEA Customer Support (through BEA WebSUPPORT at www.bea.com) for assistance.

About the AS1 and AS2 Standards

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. To learn more about the AS1 and AS2 standards, see “Electronic Data Interchange-Internet Integration (ediint)” at the following URL: <http://www.ietf.org/html.charters/ediint-charter.html>.

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.

Comparison of AS1 (SMTP/e-mail) and AS2 (HTTPS)

The following table compares features of AS1 and AS2:

Table D-1 Comparison Between AS1 and AS2

AS1	AS2
Asynchronous SMTP server connection with your interchange partner	Synchronous communication
Supports digital signatures	Supports digital signatures
Non-repudiating	Non-repudiating
Supports EDIINT AS1 compatible applications	Supports EDIINT AS2 compatible applications
Transaction-time based on E-mail server reaction	Ease of integration with back-end systems
File size limited to E-mail capacity	Unlike AS1, there is no externally imposed file-size limitation as the connection is direct and immediate
	Persistent connectivity to the Internet

Adapter Support for AS1 and AS2

The BEA WebLogic Adapter for SWIFT supplies matching interoperability support for EDIINT AS1 and AS2, as well as support for the underlying 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 might use either SMTP (S/MIME) protocol or HTTPS, regardless of source protocol.

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

Adapter Support for AS1 and AS2 Communications

Index

A

- ackagent attribute A-2
- application views
 - adding events to 3-12
 - events, adding 3-12
 - final configuration tasks 3-20
 - overview of defining 3-3
 - preparing to define 3-2
 - services, adding 3-6
 - services, testing 3-23
 - testing events manually 3-28
 - testing events using a service 3-26
 - testing services 3-23
- AS1 specification D-1
- AS2 specification D-1
- attribute parameters A-5
- attributes A-23
 - cd A-23
 - class A-3
 - errorcode A-15, A-23
 - format A-16
 - line A-15
 - max A-15
 - method A-3
 - min A-15
 - name A-3
 - stag A-3
 - tag A-3
 - tagset A-23
 - type A-19
 - usage A-3
- auditing events 3-21

B

- BEA WebLogic Adapter for SWIFT, overview of 1-1
- BEA_SWIFT_8_1.manifest.zip file 2-8
- benefits of SWIFT adapter 1-3

C

- cd attribute A-23
- checkAddition rule A-26
- checkCD rule A-23
- checkChildSequence rule A-25
- checkNodes rule A-25
- checkRelation rule A-27
- checkRepetitive rule A-24
- checkSegment rule A-28
- checkValue rule A-21
- class attribute A-3
- customer support contact information xii

D

- document entries A-2
- Document Table Model (DTM) A-3
- document validation A-1
- DTM (Document Table Model) A-3

E

- EDIError object A-7
- EDIII D-1
- errorcode attribute A-15, A-23
- events
 - about 2-2
 - adding to application views 3-12
 - auditing 3-21
 - testing 3-26
 - testing manually 3-28
 - testing using a service 3-26

F

format attribute A-16

G

getting started 1-8

H

hexadecimal values A-19

HTTPS D-1

I

IETF Working Group D-1

isDate rule A-12

isN rule A-11

isNotPresent rule A-15

isR rule A-12

isSWIFTDate rule A-16

isSWIFTReal rule A-16

isSWIFTTime rule A-20

isTime rule A-13

isValidISIN rule A-15

isValidMessageType rule A-20

isValidMultiLine rule A-15

isValidReference rule A-15

isValidSWIFTString rule A-17

J

Java rules A-5

L

line attribute A-15

logging 3-21

M

max attribute A-15

method attribute A-3

min attribute A-15

N

name attribute A-3

node parameters A-5

O

other attribute A-3

outbound processes 1-6

P

parameters

attribute A-5

node A-5

value A-5

product support xii

R

related information x

Rule List search codes A-8

rule lists A-8

rule violations A-6

rules A-11

tag A-3

checkAddition A-26

checkCD A-23

checkChildSequence A-25

checkNodes A-25

checkRelation A-27

checkRepetitive A-24

checkSegment A-28

checkValue A-21

isDate A-12

isN A-11

isNotPresent A-15

isR A-12

isSWIFTDate A-16

isSWIFTReal A-16

- isSWIFTTime A-20
- isTime A-13
- isValidISIN A-15
- isValidMessageType A-20
- isValidMultiLine A-15
- isValidReference A-15
- isValidSWIFTString A-17
- tag A-3
- tag A-3
- rules documents A-2
- rules file A-2
- rules tags A-2
- rules validation A-3

S

- schemas
 - events 2-2
 - service requests 2-2
 - service responses 2-2
- search routines in Java A-8
- service requests 2-2
- service responses 2-2
- services
 - adding to application views 3-6
 - testing 3-23
- shared.jar file 2-8
- SMTP D-1
- stag attribute A-3
- support xii
- SWIFT characters A-19
- SWIFT rule set A-14
- SWIFT X character set A-18
- SWIFT Y character set A-18
- SWIFT Z character set A-18
- SWIFTAlliance connectivity
 - iapplication server (CAS MF) C-3
 - ifile transfer (FILE and FTP) C-2
 - interactive (MQSeries) C-4
 - summary of options C-1

T

- tag attribute A-3
- tagset attribute A-23
- technical support xii
- type attribute A-19

U

- usage attribute A-3

V

- validating documents A-1
- validating rules A-3
- validation rules
 - checkAddition A-26
 - checkCD A-23
 - checkChildSequence A-25
 - checkNodes A-25
 - checkRelation A-27
 - checkRepetitive A-24
 - checkSegment A-28
 - checkValue A-21
 - isDate A-12
 - isN A-11
 - isNotPresent A-15
 - isR A-12
 - isSWIFTDate A-16
 - isSWIFTReal A-16
 - isSWIFTTime A-20
 - isTime A-13
 - isValidISIN A-15
 - isValidMessageType A-20
 - isValidMultiLine A-15
 - isValidReference A-15
 - isValidSWIFTString A-17
- value parameters A-5

W

- writing rule lists A-8

writing rules in Java A-5

X

XD.RULE_SYNTAX error A-7

XD.RULE_VIOLATION error A-7

XDException file A-6

XDRuleList class A-8