



BEA WebLogic Adapter for HIPAA

User Guide

Copyright

Copyright © 2002 BEA Systems, Inc. All Rights Reserved.

Copyright © 2002 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 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.

BEA WebLogic Adapter for HIPAA User Guide

Part Number	Date
N/A	November 2002

Table of Contents

About This Document

What You Need to Know	viii
Related Information.....	viii
Contact Us!	ix
Documentation Conventions	x

1. Introduction to HIPAA

The BEA WebLogic Adapter for HIPAA	1-3
WebLogic Integration and HIPAA.....	1-4
Document Conversion.....	1-5
The BEA WebLogic Adapter for HIPAA Toolkit	1-8
Rules Files	1-10
EDI to XML Transformation.....	1-10
Event Transport Protocol Options.....	1-10
HIPAA Validation Processing.....	1-11
Configuration.....	1-12

2. Metadata, Schemas, and Repositories

Understanding Metadata.....	2-2
Schemas and Repositories	2-3
The Repository Manifest	2-4
Message Schemas, Rules, and Code Sets	2-5
Samples File	2-5

3. Defining an Application View for the BEA WebLogic Adapter for HIPAA

Metadata	3-2
----------------	-----

Creating a New Application View.....	3-3
4. Service and Event Configuration	
Adding a Service to an Application View	4-2
Adding an Event to an Application View.....	4-8
Deploying an Application View	4-19
Testing Services and Events	4-21
5. BEA WebLogic Adapter for HIPAA Integration Using Business Process Management	
Business Process Management Functionality.....	5-1
6. Writing and Editing Rule Specification Files	
Rule Specification Files.....	6-2
Example: Creating a Simple Rule Specification File.....	6-2
Reference: Syntax for Writing Rules	6-3
Built-in HIPAA Rules	6-5
Condition Designator (cd=).....	6-5
Reference: Supported Designators	6-6
If/Then Date Format Rules.....	6-7
HIPAA Rule Set	6-7
checkDTM.....	6-8
checkDTP	6-8
checkDMG	6-9
checkQTY.....	6-9
checkCD	6-10
isN.....	6-10
isR.....	6-11
isDate	6-11
isTime	6-12
isFDATE.....	6-12
checkLen.....	6-13
checkUsage.....	6-13
isCDate	6-15
checkList.....	6-16

Example: Resolving a Checklist File Alias in the Custom Dictionary	6-16
checkEQ	6-17
segXO	6-17
compositeIntegrity	6-18
relationalIntegrity	6-18
loopSegCount	6-19
balance	6-19
tranBal	6-20
Rules In Java.....	6-20
Example: Writing Rules in Java.....	6-21
Writing Rule Search Routines in Java.....	6-23
Example: Loading a Java File	6-24

7. Functional Acknowledgement Handling

Acknowledgement Processing.....	7-2
Documents, Validation, and Acknowledgement.....	7-3
Acknowledgement Agent	7-5
Acknowledgement Message Handling.....	7-6
Creating an Acknowledgement Event.....	7-7
Testing Acknowledgement Message Handling.....	7-14



About This Document

The *BEA WebLogic Adapter for HIPAA User Guide* is organized as follows:

- [Chapter 1, “Introduction to HIPAA,”](#) introduces the BEA WebLogic Adapter for HIPAA, describes its features, and gives an overview of how it works.
- [Chapter 2, “Metadata, Schemas, and Repositories,”](#) describes metadata, how to name a schema repository and the schema manifest, how to create a schema, how to store directory and template files for transformations.
- [Chapter 3, “Defining an Application View for the BEA WebLogic Adapter for HIPAA,”](#) describes how metadata is used and how application views are created.
- [Chapter 4, “Service and Event Configuration,”](#) describes how to add services and events to application views.
- [Chapter 5, “BEA WebLogic Adapter for HIPAA Integration Using Business Process Management,”](#) describes how events are incorporated into workflow design.
- [Chapter 6, “Writing and Editing Rule Specification Files,”](#) describes how rules files work with the validation engine and how these files can be customized to suit an enterprise’s needs.
- [Chapter 7, “Functional Acknowledgement Handling,”](#) describes the process of acknowledging a document after it has passed through validation.

What You Need to Know

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

Related Information

The following documents provide additional information for the associated software components:

- *BEA WebLogic Adapter for HIPAA Installation and Configuration Guide*
- *BEA WebLogic Adapter for HIPAA Release Notes*
- *BEA Application Explorer Installation Guide*
- BEA WebLogic Server installation and user documentation, which is available at the following URL:

http://edocs.bea.com/more_wls.html

- BEA WebLogic Integration installation and user documentation, which is available at the following URL:

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

Contact Us!

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

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

If you have any questions about this version of the BEA WebLogic Adapter for HIPAA, or if you have problems using the BEA WebLogic Adapter for HIPAA, 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.
monospace text	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard. <i>Examples:</i> #include <iostream.h> void main () the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float
monospace boldface text	Identifies significant words in code. <i>Example:</i> void commit ()
<i>monospace italic text</i>	Identifies variables in code. <i>Example:</i> String <i>expr</i>
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. <i>Examples:</i> LPT1 SIGNON OR

Convention	Item
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
[]	Indicates optional items in a syntax line. The brackets themselves should never be typed. <i>Example:</i> buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...
	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
...	Indicates one of the following in a command line: <ul style="list-style-type: none">■ That an argument can be repeated several times in a command line■ That the statement omits additional optional arguments■ That you can enter additional parameters, values, or other information The ellipsis itself should never be typed. <i>Example:</i> buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...
. . . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.



1 Introduction to HIPAA

This section introduces the BEA WebLogic Adapter for HIPAA, describes its features, and gives an overview of how it works. It includes the following topics:

- [The BEA WebLogic Adapter for HIPAA](#)
- [WebLogic Integration and HIPAA](#)
- [Document Conversion](#)
- [The BEA WebLogic Adapter for HIPAA Toolkit](#)
- [Rules Files](#)
- [EDI to XML Transformation](#)
- [HIPAA Validation Processing](#)
- [Configuration](#)

The Health Insurance Portability and Accountability Act (HIPAA) was enacted in 1996 and includes provisions for the following improvements in the US health care system nationally:

- Assurance of the portability of health plan coverage
- Limitations on how pre-existing condition restrictions can be applied
- Strict protection of a patient's right to privacy and confidentiality
- Administrative simplification to reduce non-clinical health care operating costs

The impetus behind the fourth requirement was no less than the federal government's objective of balancing the federal budget and the reduction of its spending deficit. HIPAA helps support this macroeconomic objective, theoretically, by mandating the use of standards whenever health care entities send automated transactions to one another. That includes any entity doing business with HCFA (Health Care Financing Administration) health care agencies—Medicare and Medicaid. These two entities account for a large portion of the federal budget. Nearly every health care organization in the United States does business with these two government sponsors of health coverage.

Accordingly, in theory, any streamlining of business-to-business communications and related processes would provide an operational cost savings to—among others—the Federal Government's health care branches. This is but one factor that has catalyzed a push for HIPAA enactment and enforcement.

To make HIPAA work, the secretary of the Department of Health and Human Services was required to adopt standards to support the Electronic Data Interchange (EDI) of administrative and financial health care transactions primarily between health care providers and plans. The choices were:

- ANSI X12 for EDI in health care
- October 1997 ASC X12 Standards Version 4, Release 1, Sub-release 0 (004010)
- NCPDP for EDI Specific to Retail Pharmacies
- NCPDP, Real-Time Transactions Version 5.1
- NCPDP, Batch Transactions Version 1.0

Various medical and nonmedical code sets would be adopted for use in conjunction with these standards.

The provisions for administrative simplification include the use of national identifiers for the following key entities in health care:

- Payers
- Employers
- Providers
- Patients

The nomenclature for these identifiers has yet to be agreed upon.

The provision for protection of a patient's right to privacy and confidentiality has been only vaguely defined in terms of precisely how that should be achieved when information technology (IT) is involved.

HIPAA mandates present both business and information technology challenges. The BEA WebLogic Adapter for HIPAA is designed to help meet the challenges of the HIPAA mandate, by enabling accelerated integration of IT assets, thus allowing health care enterprises to realize the business benefits of the HIPAA mandate.

The BEA WebLogic Adapter for HIPAA

The BEA WebLogic Adapter for HIPAA enables fast integration of HIPAA EDI transactions into your existing environment. This kit enables developers to parse, transform, validate, store, and integrate health care information into the existing enterprise and pass information electronically to partners in HIPAA mandated form. To enable fast integration, BEA not only has supplied a parser for the HIPAA EDI documents, but also has supplied pre-built templates that enable developers to convert EDI documents to XML form or XML documents into EDI form.

The schemas required for mapping to or from the EDI format and dictionaries are used to describe the 12 HIPAA EDI transactions. These files are used by transformation templates (.xch suffix) that map the conversion to and from XML. Rules, schemas, dictionaries, and transformation templates can be customized, if the need arises. If a dictionary changes, the new corresponding schema and transformation template can be updated to reflect the change in the document by running a simple procedure. The toolkit consists of three major components allowing integration of HIPAA into your enterprise:

- Optional protocol support for File, FTP, MQ Series, and HTTP
- HIPAA toolkit containing:
 - HIPAA ASC X12N 4010 dictionaries
 - XML Schemas
 - Rules files and transformation templates
- Event and service adapters

The adapter provides pre-packaged support for the HIPAA standard documents, 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.

WebLogic Integration and HIPAA

WebLogic Integration is the hub of the BEA WebLogic Adapter for HIPAA. This service performs the runtime transformation of the HIPAA transaction sets. The adapter deploys and references transformation templates (.xchs) at run time. The adapter also applies the pre-built rules to the transformed EDI document and builds the functional acknowledgement or 997 documents.

Multiple protocol (optional adapter features) are configurable and transformation is normally applied per event or service, that is, an event or service for a specific HIPAA document type. The document read or write and conversion operations are manipulated at the document level by simple and intuitive JSP pages in the WebLogic Integration Application View Console.

When applying a transformation in a WebLogic Integration solution (using the BEA WebLogic Adapter for HIPAA service and event application views), the EDI document is sent to the workflow as XML for business processing.

Document Conversion

The BEA WebLogic Adapter for HIPAA runs an application view (service or event), which is configured to transform HIPAA EDI documents to XML documents (and vice versa), and validate the HIPAA documents (based on the published implementation guides). The process involves the transformation of the incoming EDI document, applying all the mapping rules that have been created at design time, using the application view definition. After the document is transformed to XML, the level 1-5 validation tests are performed. The rules engine uses a rule file (supplied for each transaction) that applies rules as per the implementation guide for each transaction. After validation, a functional acknowledgement is created and can be routed back to the originator of the transaction.

The process for converting XML to HIPAA is almost the reverse process, with an exception if the XML document is built incorrectly prior to transformation to EDI format.

The following diagrams show the steps for document conversion from:

1. EDI to XML
2. XML to EDI

Figure 1-1 Converting an EDI Document to an XML Document

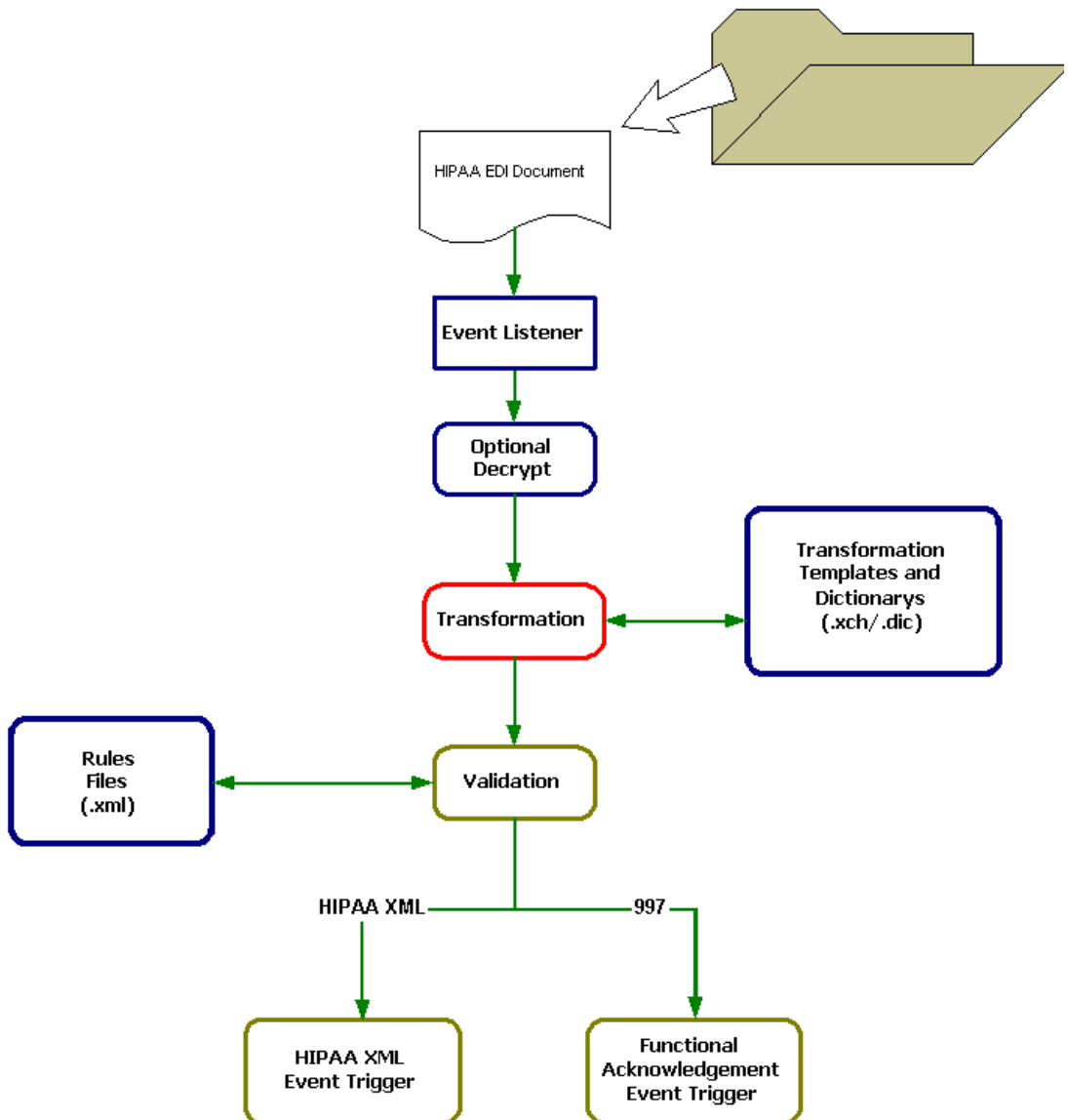
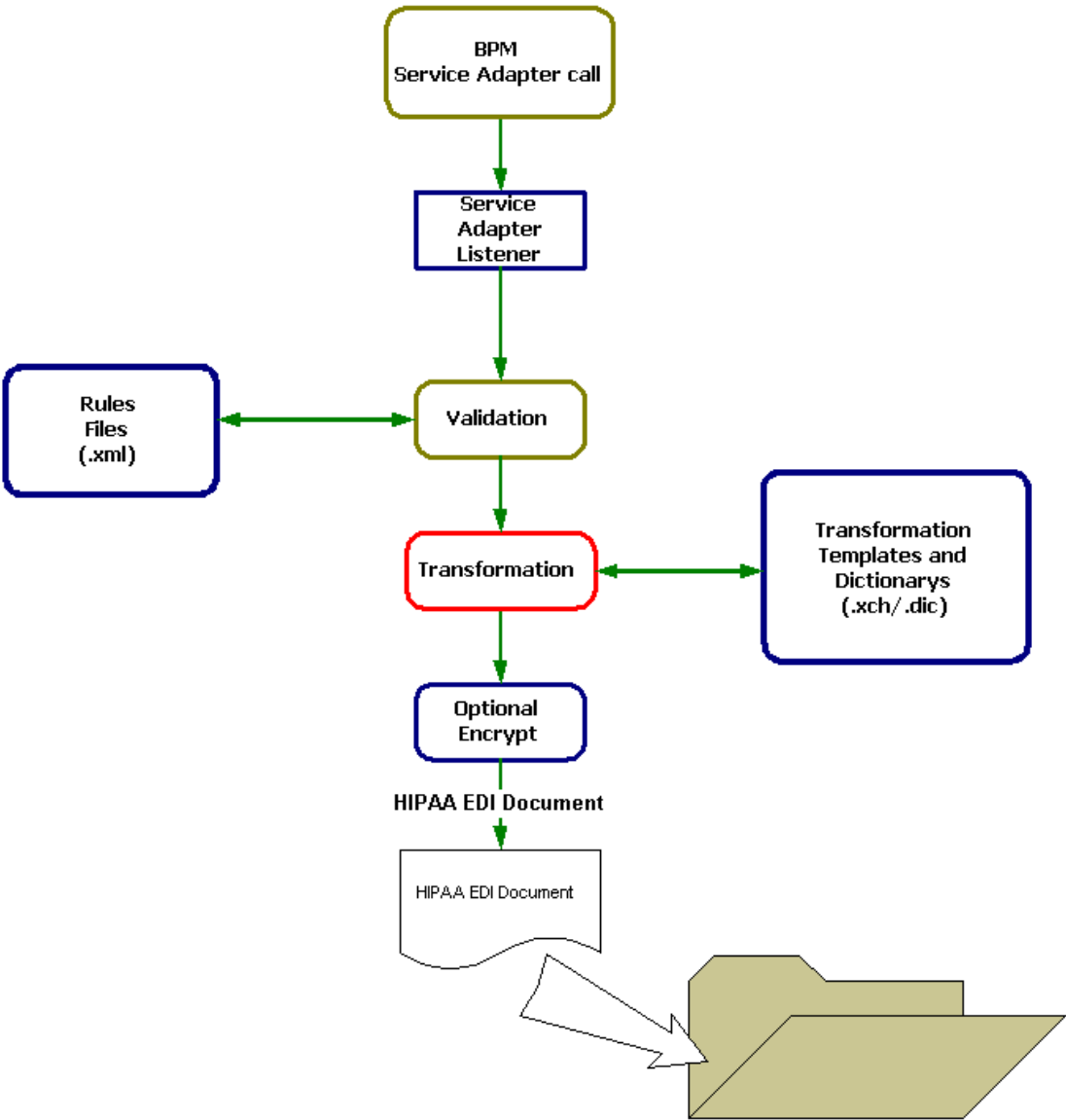


Figure 1-2 Converting an XML Document to an EDI Document



The BEA WebLogic Adapter for HIPAA comes with a WebLogic-based JSP Console page that allows authorized users to manage the adapter, protocol options, validation, rules, and other configuration tasks.

The BEA WebLogic Adapter for HIPAA Toolkit

The BEA WebLogic Adapter for HIPAA toolkit includes prebuilt XML to EDI and EDI to XML templates for all of the 4010 HIPAA transaction sets.

Figure 1-3 Documents Supplied for XML to EDI and EDI to XML Translation

Doc #	Description	Templates (.xch)	Rules files (.xml)	Sample Files	Schemas (.xsd)
270	Health Care Eligibility Inquiry	270_XML_HIPAA 270_HIPAA_XML	HIPAA270rules	270_XMLStruc.xml 270_eligibility_request.data	270
271	Health Care Eligibility Response	271_XML_HIPAA 271_HIPAA_XML	HIPAA271rules	271_XMLStruc.xml 270_eligibility_response.data	271
276	Health Care Claim Status Request	276_XML_HIPAA 276_HIPAA_XML	HIPAA276rules	276_XMLStruc.xml 276_claimStatus_request.data	276
277	Health Care Claim Status Response	277_XML_HIPAA 277_HIPAA_XML	HIPAA277rules	277_XMLStruc.xml 277_claimStatus_response.data	277
278Req	Health Care Services Review - Request for Review	278Req_XML_HIPAA 278Req_HIPAA_XML	HIPAA278Request_rules	278Req_XMLStruc.xml 278_review_request.data	278Req
278Res	Health Care Services Review - Response	278Res_XML_HIPAA 278Res_HIPAA_XML	HIPAA278Response_rules	278Res_XMLStruc.xml 278_review_response.data	278Res
820	Payroll Deducted and Other Group Premium Payment for Insurance Products	820_XML_HIPAA 820_HIPAA_XML	HIPAA820rules	820_XMLStruc.xml 820_premiumPayment.data	820
834	Benefit Enrollment and Maintenance	834_XML_HIPAA 834_HIPAA_XML	HIPAA834rules	834_XMLStruc.xml 834_benefitEnrollment.data	834
835	Health Care Claim Payment/Advice	835_XML_HIPAA 835_HIPAA_XML	HIPAA835rules	835_XMLStruc.xml 835_remittance.data	835
837I	Health Care Claim - Institutional	837Ins_XML_HIPAA 837Ins_HIPAA_XML	HIPAA837Irules	837Ins_XMLStruc.xml 837_inst.data	837Ins
837D	Health Care Claim - Dental	837Den_XML_HIPAA 837Den_HIPAA_XML	HIPAA837Drules	837Den_XMLStruc.xml 837_dental.data	837Den
837P	Health Care Claim - Professional	837Pro_XML_HIPAA 837Pro_HIPAA_XML	HIPAA837Prules	837Pro_XMLStruc.xml 837_Professional.data	837Pro.xsd

These documents must be stored under the %wldomain% in the following directory structure:

- HIPAA\rules – Validation files (.xml)
- HIPAA\xsd – Schemas for XML structure files (.xsd)
- HIPAA\dic - Dictionary files (.dic)
- HIPAA\codesets - codesets used for lookups by rules processing (.txt)
- HIPAA\samples – Sample HIPAA EDI files and XML equivalents
- HIPAA\templates – Transformation templates (.xch)

All of these documents are required to successfully transform and validate HIPAA EDI documents (both to and from XML format). The BEA WebLogic Adapter for HIPAA is pre-configured to apply transformation and validation based on the templates (.xch files) and schemas (.xsd files) supplied.

The schemas are referenced in the supplied manifest.xml document that must be located in the EIS folder required for configuration of events and services in the Application View Console.

Rules Files

The rules files are prebuilt to apply the HIPAA mandated level 1–6 rules. Level 1–5 rules are pre-configured for any other rules, and custom rules can be built and added to existing rules files. Rules are applied after the EDI document has been converted to XML. They are associated with the converted document by the root tag of the document. The BEA WebLogic Adapter for HIPAA is pre-configured to apply these rules to the relevant document.

EDI to XML Transformation

Prebuilt templates are supplied to transform the HIPAA EDI documents to XML and vice versa. Schemas and sample XML files are also supplied to assist you in the development of transformations from non-XML format to HIPAA mandated EDI form.

The workflow can be used to build templates of business processes to convert to application-specific XML or EDI form.

Event Transport Protocol Options

The WebLogic JSP Console application view pages are used to configure the receipt (based on incoming protocol) and transformation of a HIPAA document.

Functional Acknowledgements (997) are created automatically (for the inbound or event processing) and are then passed to WebLogic Integration for use or routing through the workflow service.

HIPAA Validation Processing

The BEA WebLogic Adapter for HIPAA provides the capability to validate the structure and content of incoming and outgoing documents. This goes beyond simply checking the structure that may be expressed in a schema or DTD. It includes such checks as content values, complex conditional dependencies of elements, and the balancing of values.

Some industries have stringent rules regarding the format of exchanged documents and the Validation Engine helps to meet these requirements without writing customized code. An example is the health care field, where Health Insurance Portability and Accountability Act (HIPAA) compliance involves validating that each document exchanged meets every single rule defined in the Implementation Guide. Financial documents, such as SWIFT (Society Worldwide for Interbank Financial Transactions) represent another area where content validation is required.

Validation is accomplished by using the BEA WebLogic Adapter for HIPAA, which includes a set of built-in validation rules. This set of rules provides complete coverage for validating documents related to the adapter type (for example, HIPAA and SWIFT). These rules are then invoked as defined by a *Rule Specification file*. A Rule Specification file indicates exactly which built-in rule to invoke and to which elements or segments in a document to apply the rule. These Rule Specification files are also supplied for HIPAA and SWIFT, so validation is usually just a matter of configuration.

The Rule Specifications are stored in XML format files that are freely accessible in the BEA WebLogic Adapter for HIPAA directory structure. Keeping each rule in an external file facilitates the maintenance of existing rules and provides an easy way to add new ones.

It is also possible to create new rules by writing custom Java code.

Configuration

The transformation and validation of HIPAA documents relies on three reference files that are used in the process. Dictionaries (`.dic`) files describe the structure of the EDI format file. Transformation templates (`.xch`) are used in the conversion to or from XML form. Rules files (`.xml`) apply pre-built rules, based in the ASC X12N 4010 Implementation guides. All of these files are customizable. If the dictionary is changed, a utility can be run to amend the `.xch` files and create new schemas accordingly.

2 Metadata, Schemas, and Repositories

This section explains how metadata for your enterprise information system (EIS) is described and used. After the metadata for your EIS is described, you can create and deploy application views using the WebLogic Application View Console.

This section includes the following topics:

- [Understanding Metadata](#)
- [Schemas and Repositories](#)
- [The Repository Manifest](#)
- [Message Schemas, Rules, and Code Sets](#)

Understanding Metadata

When you define an application view, you are creating an XML-based interface between WebLogic Integration and an enterprise information system (EIS) or application within your enterprise. The BEA WebLogic Adapter for HIPAA is used to define a file-based interface to applications within and outside of the enterprise. Many applications or information systems use file systems to store and share data. These files contain information required by other applications, and this information can be fed via the BEA WebLogic Adapter for HIPAA.

The BEA WebLogic Adapter for HIPAA can exploit multiple protocols to receive or emit HIPAA messages. The adapter facilitates conversion to and from XML. XML can be used within WebLogic Integration in business process workflow. WebLogic Integration requires that XML versions of HIPAA documents match schemas (provided) so that mapping for events or services can be enabled. Events and services use this information to validate the documents (either post-HIPAA to XML transformation or pre-XML to HIPAA transformation). The reference for an event or service, to a schema or set of schemas, is provided in the `manifest.xml` file.

In the `manifest.xml` file, the `schemaref` tag indicates a schema instance and includes the name to select from the schema drop-down list in the event and service JSP console. The `request` and `response` tags relate to a service. A `request` tag represents the XML being sent to the service. A `response` tag represents the response XML document received from the service request. `Event` relates to the schema for the incoming document (after conversion) to the event listener. Schemas and the related manifest files are stored in a folder or directory in the local file system that is referred to as the EIS repository. The repository location is required when creating an application view from which events and services are configured. The EIS is set up automatically (and is populated with the required schemas and manifest) when an application view is defined.

Events are triggers to workflows. When a particular file arrives at a location, an event can be triggered to read and convert, if necessary, to the XML format that conforms to a particular schema, which then initiates a flow. Services are called from the workflow to perform supported operations.

Schemas and Repositories

You describe all the documents entering and exiting your WebLogic Integration system using W3C XML schemas. These schemas describe each event arriving to and propagating out of an event, and each request sent to and each response received from a service. There is one schema for each event and two for each service (one for the request, one for the response). The schemas are usually stored in files with an `.xsd` extension.

Use the WebLogic Integration Application View Console to access events and services and to assign a schema to each event, request, and response. Assign each application view to a schema repository; several application views can be assigned to the same repository.

BEA WebLogic Adapters all make use of a schema repository to store their schema information and present it to the WebLogic Application View Console. The schema repository is a directory containing:

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

The Repository Manifest

Each schema repository has a manifest that describes the repository and its schemas. This repository manifest is stored as an XML file named `manifest.xml`.

The following is an example of a sample manifest file showing relationships between events and services and their related schemas.

The manifest file relates documents (through their schemas) to services and events. The manifest exposes schema references to the event relating the required document (via the root tag) to the corresponding schema. Schemas and manifests are stored in the same directory, the repository root of the EIS. The following is an example of the a manifest file with a description of the elements.

Listing 2-1 Sample Manifest File

```
<manifest>
  <connection/>
  <schemaref name=HIPAA270>
    <request root=HIPAA270 file=HIPAA270.xsd/>
    <response root=emitStatus file=fileEmit.xsd/>
    <event root=HIPAA270 file=HIPAA270.xsd/>
  </schemaref>
</manifest>
```

The manifest has a schema reference section, named `schemaref`. The schema reference name is displayed in the schema drop-down list in the Add Service and Add Event windows in the WebLogic Integration Application View Console. This sample manifest has three schema references or `schemaref` tags; one for services only, one for events only, and one for a combination of services and events.

Events require only one schema, defined by the event tag. This relates the root tag of an XML document to a schema in the EIS repository. For services, two schemas are required: one for the document being passed to the service, represented by the request tag, and one for the expected response document received from the service operation, represented by the response tag.

Message Schemas, Rules, and Code Sets

The BEA WebLogic Adapter for HIPAA supports the exchange of XML and non-XML messages with WebLogic Integration. In addition to converting/creating the HIPAA documents, the BEA WebLogic Adapter for HIPAA validates the structure and content. To enable this type of conversion and validation, message schemas, rules files, and code sets (for lookups) are placed in the EIS repository that is created when creating an application view. The root for the EIS is based on your version of HIPAA (4010). Dictionaries, stored in the `<EIS_Root>/4010/dictionaries` directory, describe the HIPAA messages. Rules files, stored in the `<EIS_Root>/4010/rules` directory, initiate validation by applying rules to the data via xpath reference to specific tags. Standard rules are supplied to check format (to validate data types) and check data against code set files stored in the `<EIS_Root>/4010/rules/code sets` directory. When an application view is created, the directory structures are created, and the required metadata are stored in the the directories.

Samples File

Supplied with the BEA WebLogic Adapter for HIPAA are sample files (`.xml` and `.edi` format) that can be used to help test that your environment is correctly set up and working.

3 Defining an Application View for the BEA WebLogic Adapter for HIPAA

This section describes how metadata is used and how application views are created. It includes the following topics:

- [Metadata](#)
- [Creating a New Application View](#)

Metadata

When you define an application view, you are creating an XML-based interface between WebLogic Server and a particular Enterprise Information System (EIS) application within your enterprise. In the case of the BEA WebLogic Adapter for HIPAA, this is a set of file schemas that your applications must create or respond to.

Each HIPAA document has an XML equivalent and for each of these HIPAA XML documents there is a corresponding schema. Event and service adapters must be aware of these schemas in order for them to process the document. The adapter uses a manifest file (`manifest.xml`) to relate schemas to their XML counterparts. The manifest file is located in the EIS repository that is selected when creating an application view. For more information about establishing your EIS repository, see [Chapter 2, “Metadata, Schemas, and Repositories.”](#)

Creating a New Application View

1. Open the Application View Console, which is found at the following location:

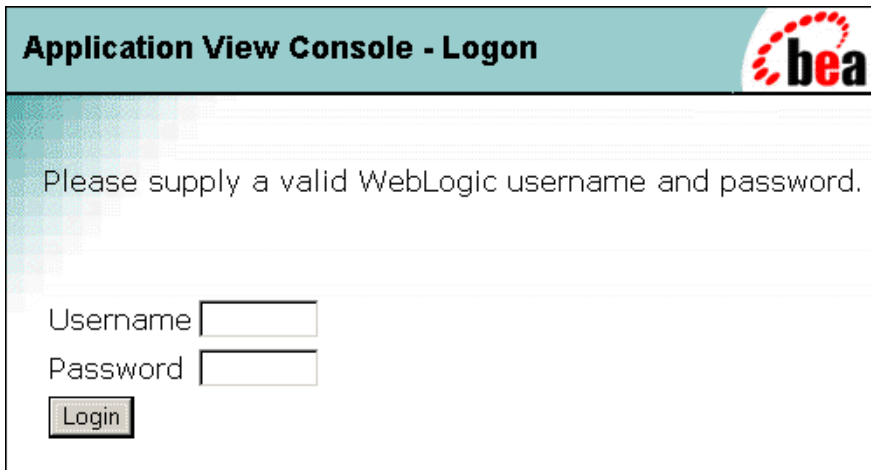
`http://host:port/wlai`

Here, *host* is the TCP/IP address or DNS name where WebLogic Integration Server is installed, and *port* is the socket on which the server is listening. The default port at the time of installation is 7001.

For more information, see “Logging On to the WebLogic Integration Application View Console” in “Defining an Application View” in *Using Application Integration*:

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

Figure 3-1 Application View Console - Logon



Application View Console - Logon

Please supply a valid WebLogic username and password.

Username

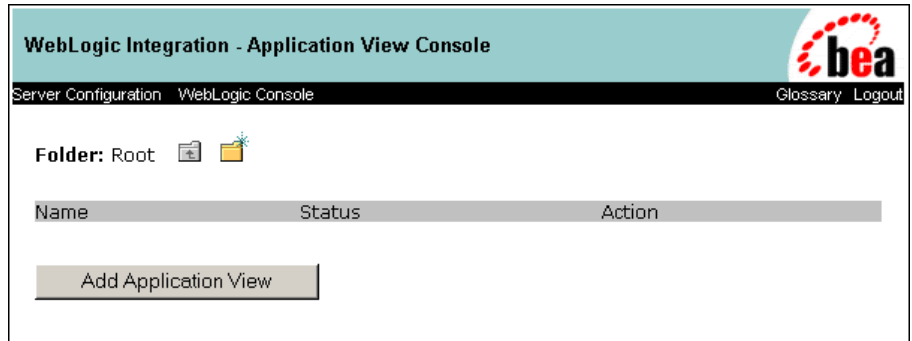
Password

Note: If the user name is not `system`, it must be included in the adapter group. For more information on adding the administrative server user name to the adapter group, see the *BEA WebLogic Adapter for HIPAA Installation and Configuration Guide*.

3 Defining an Application View for the BEA WebLogic Adapter for HIPAA

2. Enter a user name and password and click Login. The Application View Console opens.

Figure 3-2 Application View Console



3. Click Add Application View to create a new application view for the appropriate adapter.
4. An application view enables a set of business processes for this adapter's target EIS application. For more detailed information, see "Defining an Application View" in *Using Application Integration*:
 - For WebLogic Integration 7.0, see <http://edocs.bea.com/wli/docs70/aiuser/2usrdef.htm>
 - For WebLogic Integration 2.1, see http://edocs.bea.com/wlintegration/v2_1sp/aiuser/2usrdef.htm

The Define New Application View window opens.

Figure 3-3 Define New Application View Window

Define New Application View

Glossary Logout

This page allows you to define a new application view

Folder: [Root](#)

Application View Name:*

Description:

Associated Adapter:

5. In the Application View Name field, enter a name.
The name should describe the set of functions performed by this application.
Each application view name must be unique to its adapter. Valid characters are a-z, A-Z, 0-9, and _ (underscore).
6. In the Description field, enter any relevant notes. These notes are viewed by users when they use this application view in workflows using business process management functionality.
7. From the Associated Adapter list, select the particular adapter to use to create this application view.
8. Click OK.

The Configure Connection Parameters page opens.

Figure 3-4 Configure Connection Parameters Window

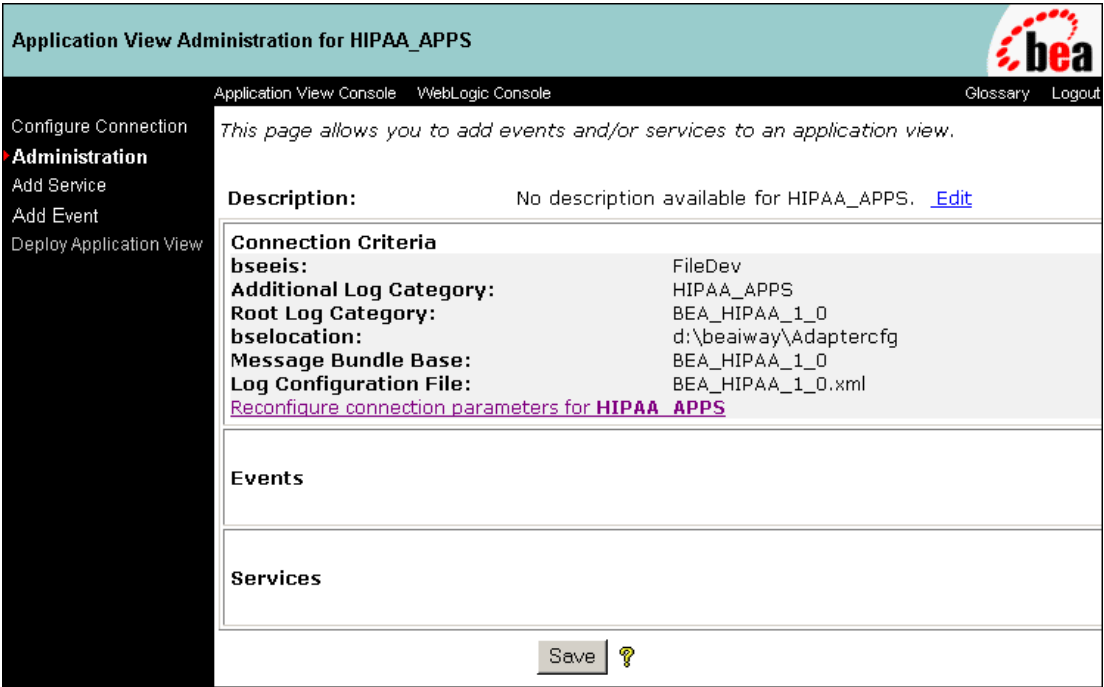
The screenshot shows the 'Configure Connection Parameters' window. The title bar includes the BEA logo and the text 'Configure Connection Parameters'. Below the title bar, there are tabs for 'Application View Console' and 'WebLogic Console', and links for 'Glossary' and 'Logout'. On the left, a navigation pane lists 'Configure Connection' (selected), 'Administration', 'Add Service', 'Add Event', and 'Deploy Application View'. The main content area has a heading 'On this page, you supply parameters to connect to your EIS'. Below this, a paragraph explains that the BEA Application Explorer generates schema information for a session stored at a location that must be known to the general adapter. It states that a session can support multiple connections. Another paragraph instructs the user to enter the 'session path' location and click the pulldown arrow for the 'connection name', which will display a selection list of valid connections. The form contains two input fields: 'Session path*' with the value 'd:\beaiway\Adaptercdg' and 'Connection name*' with a dropdown menu showing 'FileDev'. A 'Connect to EIS' button is located at the bottom of the form.

The metadata for the HIPAA EIS is contained in the `manifest.xml` and schemas provided with this product.

For more information on the session path and connection name, see [Chapter 2, “Metadata, Schemas, and Repositories.”](#)

9. Enter the root and select the connection name (the application folder containing schemas and the manifest file) from the drop-down list.
10. Click Connect to EIS to view the Application View Administration window.

Figure 3-5 Application View Administration Window



3 *Defining an Application View for the BEA WebLogic Adapter for HIPAA*

4 Service and Event Configuration

This section describes the configuration of services and events and shows the setting options available for enabling the Enterprise Information System (EIS). It includes the following topics:

- [Adding a Service to an Application View](#)
- [Adding an Event to an Application View](#)
- [Deploying an Application View](#)
- [Testing Services and Events](#)

After the EAR file, metadata repository, and application views are defined and available to WebLogic Integration, services and events can be added to the newly created application view.

For information about the EAR file, see the *BEA WebLogic Adapter for HIPAA Installation and Configuration Guide*. For information about establishing your EIS repository, see [Chapter 2, “Metadata, Schemas, and Repositories.”](#)

For information about defining application views, see [Chapter 3, “Defining an Application View for the BEA WebLogic Adapter for HIPAA.”](#)

Adding a Service to an Application View

After you create and configure an application view, add services that support the application's functions.

1. While the application view is open (and undeployed), select Administration from the Configure Connection list in the left pane. The Application View Administration window opens.

Figure 4-1 Application View Administration Window

Application View Administration for HIPAA_APP

Application View Console WebLogic Console Glossary Logout

Configure Connection
Administration
 Add Service
 Add Event
 Deploy Application View

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

Description: No description available for HIPAA_APP. [Edit](#)

Connection Criteria

bseis:	FileDev
Additional Log Category:	HIPAA_APP
Root Log Category:	BEA_HIPAA_1_0
bselocation:	d:\beaiway\Adaptercfg
Message Bundle Base:	BEA_HIPAA_1_0
Log Configuration File:	BEA_HIPAA_1_0.xml

[Reconfigure connection parameters for HIPAA_APP](#)

Events

Services

Save ?

2. Click Add in the Services pane. The Add Service page opens.

Figure 4-2 Add Service Window

3. Enter a Name in the Unique Service Name field.
The name should describe the function performed by this service. Each service name must be unique to its application view. Valid characters are a-z, A-Z, 0-9, and _ (underscore).
4. Select the operation to be configured from the Select drop-down list. This list includes: File System Write, FTP Write, MQEmit, TCPEmit, and HTTP. You can configure only one operation per service.
For more information on configuration and deployment of transformation templates, see [Chapter 3, “Defining an Application View for the BEA WebLogic Adapter for HIPAA.”](#)
5. Select the protocol or format from the Transform Type drop-down list, and enter the required values (required fields are marked with an asterisk). Descriptions of the parameters for each operation are provided in the following tables:

Table 4-1 File System Write

Setting	Meaning/Properties
Transform Type	Type/Value: Drop-down list Description: Select the pre-built transformation template to apply to the expected document. One per HIPAA document. All prefixed by XML_to <DOCTYPE>.
directory* (*Required)	Type/Value: Directory Path. Description: Directory to which output messages are emitted.
output file name/mask* (*Required)	Type/Value: String Description: The output file name (can contain a '*') that expands 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.

Table 4-2 FTP Write

Setting	Meaning/Properties
Transform Type	Type/Value: Drop-down list Description: Select the prebuilt transformation template to apply to the expected document. One per HIPAA document. All prefixed by XML_to <DOCTYPE>.
Host name* (*Required)	Type/Value: String Description: FTP target system.
Port number	Type/Value: Numeric Description: FTP target system port (leave empty for FTP default).
User ID* (*Required)	Type/Value: String Description: User account ID to use when connecting to protocol host.

Table 4-2 FTP Write (Continued)

Setting	Meaning/Properties
Password* (*Required)	Type/Value: String Description: Password for user account to use when connecting to protocol host.
destination* (*Required)	Type/Value: String Description: Directory to address on FTP target system.
output file name/mask	Type/Value: String Description: The output file name (can contain a '*'), expands to a timestamp.
Retry Interval	Type/Value: Duration - xxH:xxM:xxS (for example, 1h:2m:3s = 1 hour, 2 minutes, and 3 seconds). Description: The maximum wait interval between retries when a connection fails.
Maxtries	Type/Value: String Description: Number of retries for a failed attempt to write.

Table 4-3 MQEmit

Setting	Meaning/Properties
Transform Type* (*Required)	Type/Value: Drop-down list Description: The transformation to be applied.
Queue manager* (*Required)	Type/Sample Value: String / QM_BEAHIPAA Description: Name of the MQSeries Queue Manager to be used.
Queue name* (*Required)	Type/Sample Value: String / TEST.iO Description: Queue on which request documents are received.
MQ client host	Type/Value: String Description: For MQ Client only. Host on which MQ Server is located.

4 Service and Event Configuration

Table 4-3 MQEmit (Continued)

Setting	Meaning/Properties
MQ client port	Type/Value: Integer Description: For MQ Client only. Port number to connect to an MQ Server.
MQ client channel	Type/Value: String Description: For MQ Client only. Channel between an MQ Client and MQ Server.
Polling Interval	Type/Value: String. Duration, in the format <i>nnH:nnM:nn</i> . For example, 1H:2M:3S (1 hour 2 minutes and 3 seconds) Description: The maximum wait interval between checks for new documents. The higher this value, the longer the interval, and the fewer system resources that are used. The side effect of a high value is that the worker thread will not be able to respond to a stop command. If timeout is set to 0, the listener will run once and terminate. Default is 2 seconds.

Table 4-4 TCPEmit

Setting	Meaning/Properties
Transform Type* (*Required)	Type/Value: Drop-down list Description: The transformation to be applied.
TCP port* (*Required)	Type/Sample Value: Integer / 12345 Description: TCP listening port
Allowable Client Host	Type/Sample Value: String / HIPAAServ or 123.12.23.34 Description: Host name or host address of client restricted to accessing this event adapter.
Character set encoding* (*Required)	Type/Sample Value: String / ISO-8859-1 Description: Document character set.

Table 4-5 HTTP

Setting	Meaning/Properties
Transform Type* (*Required)	Type/Value: drop down Description: Select from the transformation to be applied.
Character set encoding* (*Required)	Type/Value: String / ISO-8859-1 Description: Document character set.
Port* (*Required)	Type/Value: Integer / 12345 Description: HTTP listener port.

The following figure shows the configuration of a File System Write operation:

Figure 4-3 Add Service Window (with settings)

Add Service

Application View Console WebLogic Console

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

Unique Service Name: * HIPAA_270_OUT

Select: File System Write

Transform Type* XML_to_270

directory* d:\fileouthipaa

output file name/mask* H270*.edi

schema: 270

Add

6. Select the schema required for this service.

The schema drop-down has a selection for each HIPAA document.

The selection must correspond to the Transform Type field.

7. Click Add.

At this point, the application view can be deployed or more services or events can be configured. After the application view has been deployed, you can test the service. For more information on deploying the application view, please refer to [“Deploying an Application View” on page 4-19](#). For more information on testing the application view, see [“Testing Services and Events” on page 4-21](#).

Adding an Event to an Application View

To add events to the application view, schemas must be present and mapped to the BEA WebLogic Adapter for HIPAA EIS that is configured for the application view. For more information on creating an application view, see [Chapter 3, “Defining an Application View for the BEA WebLogic Adapter for HIPAA.”](#)

If your application is deployed, you must undeploy the application and then edit the application view.

1. From the Application View Administration window, select Add in the event section of the administration pane. See [Figure 4-1, “Application View Administration Window,” on page 4-2](#).

The Add Event window opens.

Figure 4-4 Add Event Window

Add Event

Application View Console WebLogic Console Glossary Logout

Configure Connection
Administration
Add Service
► **Add Event**
Deploy Application View

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

Unique Event Name: *

Select:

Location*	<input type="text"/>
File Suffix*	<input type="text"/>
Character Set Encoding*	<input type="text" value="ISO-8859-1"/>
Polling Interval	<input type="text" value="1"/>
Sort	<input type="checkbox"/>
Scan sub-directories	<input type="checkbox"/>
File-read limit (per scan)	<input type="text"/>
ackagent*	<input type="text" value="XDHipaaAckAgent"/>
protocol	<input type="text" value="FILE"/>
to	<input type="text"/>
Transform Type*	<input type="text" value="270_to_XML"/>

schema:

2. Enter a Unique Event Name and then configure the particular event protocol required (select the option button to activate that protocol).
3. Select the event protocol required from the Select drop-down box.
You have the option to configure an event based on one of the following protocols: File System, FTP, MQ, TCP, and HTTP.
4. Enter the required values (required fields are marked with an asterisk).
Descriptions of the parameters are provided in the following tables:

Table 4-6 File System

Setting	Meaning/Properties
Location* (*Required)	<p>Type/Value: Directory Path</p> <p>Description: Directory where input messages are received. The listener allows DOS-style file patterns for input selection. The file input section of the configuration is now a file pattern in addition to a directory. The user can enter a pattern as c:\xyz\ab*cd, WITHOUT THE SUFFIX, which is handled by the suffix list entry.</p> <p>If a pattern is used, the files are selected in order based on the suffix and then the pattern. AB?CD selects ABxCD. AB*CD selects ABxxxCD.</p>
File Suffix* (*Required)	<p>Type/Value: String</p> <p>Description: Limits input files to those with these extensions (separated by a comma). For example, in XML, do not use '!'; - means no extension.</p> <p>Note: If the file suffix is zip, the unzipped files needs to conform to the event schema or they fail. This function also works with transform configured.</p>
Character Set Encoding* (*Required)	<p>Type/Value: String</p> <p>Description: Sets the character set encoding to be used (default value ISO-8859-1-US and Western Europe).</p>
Polling interval	<p>Type/Value: Duration - xxH:xxM:xxS (for example, 1H:2M:3S = 1 hour, 2 minutes, and 3 seconds).</p> <p>Description: The maximum wait interval between checks for new documents. The higher this value, the longer the interval, and the fewer system resources that are used. The side effect of a high value is that the worker thread cannot respond to a stop command. If timeout is set to 0, the listener runs once and terminates. Default is 2 seconds.</p>
Sort	<p>Type/Value: Boolean (true or false)</p> <p>Description: Sort by arrival. If set, sort incoming documents by arrival time. Maintains sequence, but slows performance.</p>
Scan sub-directories	<p>Type/Value: Boolean (true or false)</p> <p>Description: Scans all subdirectories for documents to be processed.</p>

Table 4-6 File System (Continued)

Setting	Meaning/Properties
File-read limit (per scan)	Type/Value: Integer Description: The number of files read per sweep of the File directory location.
ackagent* (*Required)	Type/Value: String Description: The agent responsible for processing acknowledgement or non-acknowledgement of HIPAA documents. The default HIPAA acknowledgement agent will generate an empty acknowledgement document or will list all failed validation rules in the non-acknowledgement document.
protocol	Type/Value: String Description: Protocol on which to make ACK copies (Currently only FILE is supported.)
to	Type/Value: String Description: Location for ACK to be sent; for example, f:\fileout\ack.xml.
Transform Type	Type/Value: Drop-down list Description: Select the prebuilt transformation template to apply to the expected document.

Table 4-7 FTP

Setting	Meaning/Properties
User Id* (*Required)	Type/Value: String Description: User account to use when connecting to protocol host.
Password* (*Required)	Type/Value: String Description: Password for user account to use when connecting to protocol host.

Table 4-7 FTP (Continued)

Setting	Meaning/Properties
Host name* (*Required)	Type/Value: String Description: Name of host machine where listener contacts service to obtain requests from.
Location* (*Required)	Type/Value: Directory Description: Directory on FTP host to retrieve files from. You must append the file suffix (extension) to the file or files specified in the Location field. For example, you can enter a specific file such as /path/to/my/ftp/directory/myfile.xml or a group of files such as /path/to/my/ftp/directory/*.zip.
File suffix	Type/Value: String Description: This field is no longer used. You must append the file suffix to the file or files specified in the Location field.
Character Set Encoding* (*Required)	Type/Value: String Description: Sets the character set encoding to be used (default value ISO-8859-1 -US and Western Europe).
Polling interval	Type/Value: Duration - xxH:xxM:xxS (for example, 1H:2M:3S = 1 hour, 2 minutes, and 3 seconds). Description: The maximum wait interval between checks for new documents. The higher this value, the longer the interval, and the fewer system resources that are used. The side effect of a high value is that the worker thread cannot respond to a stop command. If timeout is set to 0, the listener runs once and terminates. Default is 2 seconds.
Scan sub-directories	Type/Value: Boolean (true or false) Description: Scans all subdirectories for documents to be processed.
Transform Type	Type/Value: Drop-down list Description: Select the pre-built transformation template to apply to the expected document. One per HIPAA document. All prefixed by XML_to <DOCTYPE>.

Table 4-7 FTP (Continued)

Setting	Meaning/Properties
ackagent* (*Required)	Type/Value: String Description: The agent responsible for processing acknowledgement or non-acknowledgement of HIPAA documents. The default HIPAA acknowledgement agent will generate an empty acknowledgement document or will list all failed validation rules in the non-acknowledgement document.
protocol	Type/Value: String Description: Protocol on which to make ACK copies (Currently only FILE is supported.)
to	Type/Value: String Description: Location for ACK to be sent; for example, f:\fileout\ack.xml.

Table 4-8 MQ

Setting	Meaning/Properties
Transform Type* (*Required)	Type/Sample Value: Drop-down list Description: Name of the MQSeries queue manager to be used.
Queue Manager* (*Required)	Type/Sample Value: String / QM_BEA_HIPAA Description: Name of the MQSeries queue manager to be used.
Queue Name* (*Required)	Type/Sample Value: String / TEST.iO Description: Queue on which request documents are received.
MQ Client Host	Type/Sample Value: String Description: For MQ client only. Host on which MQ Server is located.
MQ client port	Type/Value: Integer Description: For MQ client only. Port number to connect to an MQ server.

4 Service and Event Configuration

Table 4-8 MQ (Continued)

Setting	Meaning/Properties
MQ client channel	Type/Value: String Description: For MQ client only. Channel between an MQ client and MQ server.
Polling interval	Type/Sample Value: String duration in the format <i>nnH:nnM:nnS</i> / 1H:2M:3S (1 hour, 2 minutes, 3 seconds) Description: The maximum wait interval between checks for new documents. The higher this value, the longer the interval, and the fewer system resources that are used. The side effect of a high value is that the worker thread cannot respond to a stop command. If timeout is set to 0, the listener runs once and terminates. Default is 2 seconds.
ackagent* (*Required)	Type/Value: String Description: The agent responsible for processing acknowledgement or non-acknowledgement of HIPAA documents. The default HIPAA acknowledgement agent will generate an empty acknowledgement document or will list all failed validation rules in the non-acknowledgement document.
protocol	Type/Value: String Description: Protocol on which to make ACK copies (Currently only FILE is supported.)
to	Type/Value: String Description: Location for ACK to be sent; for example, f:\fileout\ack.xml.

Table 4-9 TCP

Setting	Meaning/Properties
Transform Type* (*Required)	Type/Sample Value: Drop-down list Description: The transformation to be applied.

Table 4-9 TCP (Continued)

Setting	Meaning/Properties
TCP/IP port* (*Required)	Type/Sample Value: Integer / 12345 Description: TCP listening port.
Allowable client host	Type/Sample Value: String / HIPAAServ or 123.12.23.345 Description: Host name or host address of client restricted to accessing this event adapter.
ackagent* (*Required)	Type/Value: String Description: The agent responsible for processing acknowledgement or non-acknowledgement of HIPAA documents. The default HIPAA acknowledgement agent will generate an empty acknowledgement document or will list all failed validation rules in the non-acknowledgement document.
protocol	Type/Value: String Description: Protocol on which to make ACK copies (Currently only FILE is supported.)
to	Type/Value: String Description: Location for ACK to be sent; for example, f:\fileout\ack.xml.
encoding* (*Required)	Type/Sample Value: String / ISO-8859-1 Description: Document character set.

Table 4-10 HTTP

Setting	Meaning/Properties
Transform Type* (*Required)	Type/Sample Value: Drop-down list Description: The transformation to be applied.
Character set encoding* (*Required)	Type/Sample Value: String / ISO-8859-1 Description: Document character set.
Port* (*Required)	Type/Sample Value: Integer / 12345 Description: HTTP listener port.

Table 4-10 HTTP (Continued)

Setting	Meaning/Properties
ackagent* (*Required)	Type/Value: String Description: The agent responsible for processing acknowledgement or non-acknowledgement of HIPAA documents. The default HIPAA acknowledgement agent will generate an empty acknowledgement document or will list all failed validation rules in the non-acknowledgement document.
protocol	Type/Value: String Description: Protocol on which to make ACK copies (Currently only FILE is supported.)
to	Type/Value: String Description: Location for ACK to be sent; for example, f:\fileout\ack.xml.

The following figure provides an example of a configured File System event.

Figure 4-5 Add Event Window (with settings)

Add Event

Application View Console WebLogic Console Glossary Logout

Configure Connection
Administration
Add Service
► **Add Event**
Deploy Application View

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

Unique Event Name: * HIPAA_270_IN

Select: File System ▼

Location*	d:\filein\hipaa
File Suffix*	edi
Character Set Encoding*	ISO-8859-1
Polling Interval	1
Sort	<input type="checkbox"/>
Scan sub-directories	<input type="checkbox"/>
File-read limit (per scan)	
ackagent*	XDHipaaAckAgent
protocol	FILE
to	
Transform Type*	270_to_XML ▼

schema: 270 ▼

Add

5. Select the schema required for this event. The schema drop-down list has a selection for each HIPAA document.
6. Click Add. After adding the event process, the Application View Administration window is displayed.

Figure 4-6 Application View Administration Window

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

Description: No description available for HIPAA_APP. [Edit](#)

Connection Criteria	
bseis:	FileDev
Additional Log Category:	HIPAA_APP
Root Log Category:	BEA_HIPAA_1_0
bselocation:	d:\beaiway\Adaptercfg
Log Configuration File:	BEA_HIPAA_1_0.xml
Message Bundle Base:	BEA_HIPAA_1_0

[Reconfigure connection parameters for HIPAA_APP](#)

Events	Add
HIPAA_270_IN Edit Remove Event View Summary View Event Schema	

Services	Add
-----------------	---------------------

[Continue](#) [Save](#) [?](#)

7. Click Save to save your settings.

Deploy your application view (complete with configured events and/or services) by following the steps described in [“Deploying an Application View” on page 4-19](#). Then, test your application view by following the steps described in [“Testing Services and Events” on page 4-21](#).

Deploying an Application View

You can deploy an application view when you have added at least one event or service to it. You must deploy an application view before you can test its services and events or use the application view in the WebLogic Server environment. For information on adding a service to an application view, see [“Adding a Service to an Application View” on page 4-2](#). For more information on adding an event to an application view, see [“Adding an Event to an Application View” on page 4-8](#).

Application view deployment places relevant metadata about its services and events into a run-time metadata repository. Deployment makes the application view available to other WebLogic Server clients. This means business processes can interact with the application view, and you can test the application view’s services and events.

After you configure a service or event, you can deploy your application view from the Application View Administration window.

Figure 4-7 Application View Administration Window

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

Description: No description available for HIPAA_APP. [_Edit](#)

Connection Criteria	
bseis:	FileDev
Additional Log Category:	HIPAA_APP
Root Log Category:	BEA_HIPAA_1_0
bselocation:	d:\beaiway\Adaptercfg
Log Configuration File:	BEA_HIPAA_1_0.xml
Message Bundle Base:	BEA_HIPAA_1_0
Reconfigure connection parameters for HIPAA_APP	

Events Add

HIPAA_270_IN Edit Remove Event View Summary View Event Schema

Services Add

Continue Save ?

4 Service and Event Configuration

1. From the Application View Administration window, click Continue.
The Deploy Application View window opens.

Figure 4-8 Deploy Application View Window

Deploy Application View HIPAA_APP to Server

Application View Console WebLogic Console Glossary Logout

Configure Connection Administration Add Service Add Event **Deploy Application View**

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

Required Event Parameters

Event Router URL* ?

Connection Pool Parameters

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

Minimum Pool Size*

Maximum Pool Size*

Target Fraction of Maximum Pool Size*

Allow Pool to Shrink? ☒

Log Configuration

Set the log verbosity level for this application view.

Configure Security

[Restrict Access to HIPAA_APP using J2EE Security](#)

? ☒ Deploy persistently? ? ?

Note: To enable business process management functionality for other authorized clients to asynchronously call the services (if any) of this application view, select Enable Asynchronous Service Invocation.

2. To deploy the application view, click Deploy.
The Summary for Application View page opens.

Note: You may choose to click Save and deploy the BEA WebLogic Adapter for HIPAA later.

After you create and deploy an application view, test the service and events. For more information, see [“Testing Services and Events” on page 4-21](#).

Testing Services and Events

You can test services and events after you create and deploy an application view.

To test a service:

1. In the Summary for Application View window, click Test for the configured service.
2. When the Test pane opens prompting you for the test XML, enter the required XML.
3. Click Test.

If your service has been configured correctly, you receive a response from the file emit process with a status code of “0.”

Also, you see that the file has been written to the correct location.

After you have confirmed that the file has been written correctly (in the correct format if transformation has been configured), your service adapter has been successfully configured.

You can now employ the service in business process workflows or write custom code. For more information, see “Using Application Views in the Studio” in *Using Application Integration*:

- For WebLogic Integration 7.0, see <http://edocs.bea.com/wli/docs70/aiuser/3usruse.htm>
- For WebLogic Integration 2.1, see http://edocs.bea.com/wlintegration/v2_1sp/aiuser/3usruse.htm

To test an event:

1. After you configure and deploy your event, click the Test link for the configured event in the Summary for Application View Window.
2. Test the event using the workflow (that is, the workflow triggered by an event). For more information, see [Chapter 5, “BEA WebLogic Adapter for HIPAA Integration Using Business Process Management.”](#)

5 BEA WebLogic Adapter for HIPAA Integration Using Business Process Management

This section describes how events are incorporated into workflow design. It includes the following topic.

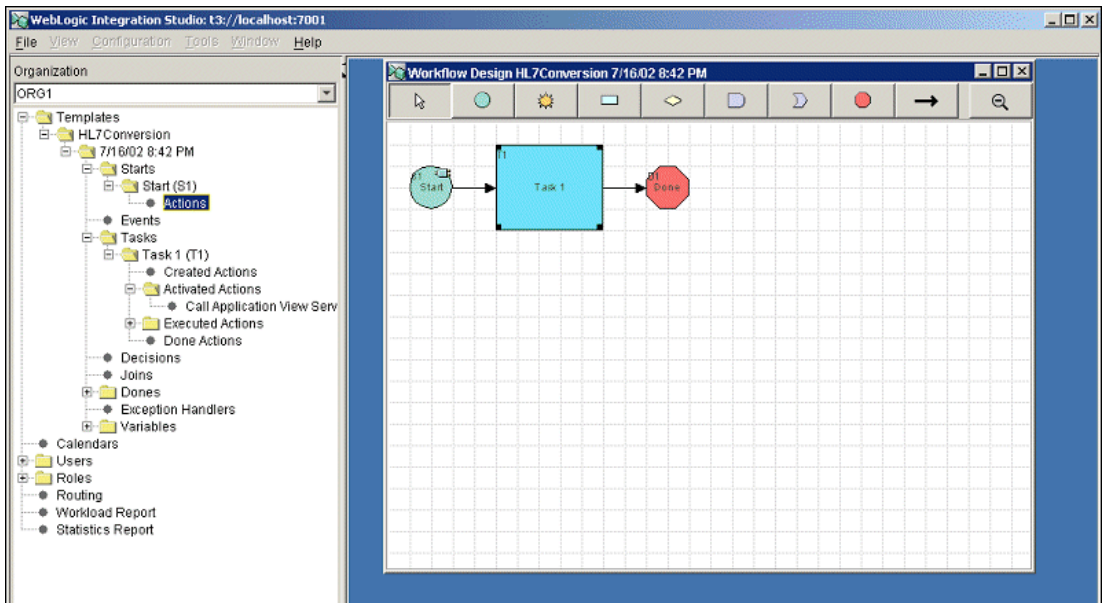
- [Business Process Management Functionality](#)

Business Process Management Functionality

After successfully creating your application view, including services and events, you can integrate it into a business process management workflow.

The following window depicts a simple workflow design triggered by an event described in [Chapter 3, “Defining an Application View for the BEA WebLogic Adapter for HIPAA.”](#) The incoming event converts a document and utilizes a service adapter to propagate the event to a file system. For example, an HIPAA 270 file taken from an FTP directory is converted to XML. After processing using business process management workflows, a HIPAA 271 response document is created and placed on a local file system. The event responds to the HIPAA 270 document being placed on the FTP site, then converts the document into XML format, and propagates it (using the workflow) to the service adapter.

Figure 5-1 Workflow Design



The following listing depicts the example file used in this process. It is an example of a 270 Eligibility Benefit Inquiry document.

Note: The first two lines in the following listing are a single line in the file.

Listing 5-1

```
ISA*00*1234567890*00*1234567890*ZZ*SUBMITTERS ID12*ZZ*RECEIVERS
ID123*010122*1253*U*00401*000000905*1*T*:~
GS*HS*SenderID*ReceiverID*20010122*1310*1*X*004010X092~
ST*270*1234~
BHT*0022*13*10001234*19990501*1319~
HL*1**20*1~
NM1*PR*2*ABCCOMPANY*****PI*842610001~
HL*2*1*21*1~
NM1*1P*1*JONES*MARCUS****SV*0202034~
REF*N5*129~
N3*55 HIGH STREET~
N4*SEATTLE*WA*00501~
PER*IC*MARYMURPHY*TE*2065551212*EX*3694*FX*2065551214~
HL*3*2*22*1~
TRN*1*93175-012547*9877281234~
NM1*IL*1*SMITH*ROBERT*B***MI*11122333301~
REF*1L*599119~
N3*29 FREMONT ST*APT# 1~
N4*PEACE*NY*10023~
DMG*D8*19430519*M~
HL*4*3*23*0~
TRN*1*93175-012547*9877281234*RADIOLOGY~
NM1*03*1*SMITH*MARYLOU~
REF*SY*003221234~
N3*29 FREMONT ST*APT# 1~
N4*PEACE*NY*10023~
DMG*D8*19781014*F~
INS*N*19~
DTP*472*D8*19990501~
EQ*81**FAM~
SE*28*1234~
GE*1*1~
IEA*1*000000905~
```

Starting with the original document, this process consists of the following steps:

1. The document is placed into an FTP directory that had a BEA WebLogic Adapter for HIPAA event configured to be triggered by the arrival of the 270 document.

The event has been pre-configured to apply an EDI to XML conversion, using the event configuration JSP.

After the adapter completes the process, the workflow then propagates the XML document onto the adapter service.

2. The workflow is configured to process the information and create the XML document required for the XML to EDI (271) transformation and write to a local file system.
3. After the service has completed its task, the emission report is returned to the workflow.

Figure 5-2 Emission Report

The screenshot displays the WebLogic Integration Studio interface. The main window shows a workflow design for 'HL7Conversion'. A 'Workflow Instances' dialog is open, showing a table of workflow instances. The table has columns for 'Workflow Label', 'Started', and 'Completed'. The data row shows a workflow labeled '2002-07-16 21:15:54.63' that started at '2002-07-16 21:15:54.63' and completed at '2002-07-16 21:15:54.65'. A 'Workflow Variables' dialog is also open, showing a table of variables. The table has columns for 'Name', 'Type', and 'Value'. The data row shows a variable named 'Address' of type 'xml' with a value of '<?xml version="1.0" encoding="UTF-8"?>...'. A 'View XML' dialog is open, showing the XML content of the variable: '<?xml version="1.0" encoding="UTF-8"?><emitStatus><protocol>FILEC/protocol</protocol><params></params><status>1</status><asp>Success</asp><timestamp>2002-07-16T24:31:41:274Z</timestamp><attempts>1</attempts><name></name></emitStatus></?xml>'. The XML is valid according to the schema '<?com.bea.wli.bpm Valid XML Template ?>'. The 'Workflow Variables' dialog has an 'Update' button and a 'View XML' button. The 'View XML' dialog has a 'Close' button.

Workflow Label	Started	Completed
2002-07-16 21:15:54.63	2002-07-16 21:15:54.63	2002-07-16 21:15:54.65

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

```
<?xml version="1.0" encoding="UTF-8"?>
<emitStatus>
  <protocol>FILEC/protocol</protocol>
  <params></params>
  <status>1</status>
  <asp>Success</asp>
  <timestamp>2002-07-16T24:31:41:274Z</timestamp>
  <attempts>1</attempts>
  <name></name>
</emitStatus>
</?xml>
<?com.bea.wli.bpm Valid XML Template ?>
```

The following file is an example of the 271 Eligibility Benefit Response file written to the output location.

```
ISA*00*1234567890*00*1234567890*ZZ*SUBMITTERS ID12*ZZ*RECEIVERS
ID123*010122*1253*U*00401*000000905*1*T*:~
GS*HB*SenderID*ReceiverID*20010122*1310*1*X*004010X092~
ST*271*1234~
BHT*0022*11*10001234*19990501*1319~
HL*1**20*1~
NM1*PR*2*ABCCOMPANY*****PI*842610001~
HL*2*1*21*1~
NM1*1P*1*JONES*MARCUS*****SV*0202034~
REF*N5*129~
HL*3*2*22*1~
TRN*2*93175-012547*9877281234~
NM1*IL*1*SMITH*ROBERT B****MI*11122333301~
REF*1L*599119~
N3*29 FREMONT ST*APT# 1~
N4*PEACE*NY*10023~
DMG*D8*19430519*M~
HL*4*3*23*0~
TRN*2*93175-012547*9877281234*RADIOLOGY~
NM1*03*1*SMITH*MARYLOU~
REF*SY*003221234~
N3*29 FREMONT ST*APT # 1~
N4*PEACE*NY*10023~
DMG*D8*19881014*F~
INS*N*19~
DTP*472*D8*19950624~
EB*1*FAM*30*GP~
EB*B**81*GP***15****Y~
EB*L~
LS*2120~
NM1*P3*1*BROWN*TOM*D**JR*SV*222333444~
PER*IC*BILLING DEPT*TE*2065556666~
LE*2120~
SE*31*1234~
GE*1*1~
IEA*1*000000905~
```

6 Writing and Editing Rule Specification Files

A complete set of validation rules is supplied with BEA format adapters such as the BEA WebLogic Adapter for HIPAA. Sometimes you may be required to edit the supplied Rule Specification files.

Validation may be applied to other types of documents by creating new Rule Specification files for them. This section provides details on constructing or editing a Rule Specification file and describes how rule files work with the validation engine and how these files can be customized to suit an enterprise's needs. It includes the following topics:

- [Rule Specification Files](#)
- [Built-in HIPAA Rules](#)
- [HIPAA Rule Set](#)
- [Rules In Java](#)
- [Writing Rule Search Routines in Java](#)

Rule Specification Files

The Rule Specification file is an XML document. One file should exist for each document type, as defined by its XML root element, to be validated. For clarity, the root element of the rule file should match the root element of the document being validated. Contained within the file are the individual *rule* elements.

A production Rule Specification likely has many `<rule>` elements—as many as are required to completely validate the entire document. Building a complete Rule Specification involves identifying each element to be validated and selecting the appropriate rule for the type of validation required. For example, the `checkList` rule validates that the element contains only values from the supplied list, and the `isFDate` rule validates that the element value has the proper date format (CCYYMMDD).

Example: Creating a Simple Rule Specification File

The structure for a simple Rule Specification file might look like this:

```
<TestDoc>
  <using class="XDHipaaRules">
    <rule tag="EL1"    method="checkList"    code="a,b,c" />
    <rule tag="EL99"   method="isFDate"     code="RD8" />
  </using>
</TestDoc>
```

The options to this file are defined as follows:

- `<TestDoc>` represents the XML type of the document to be validated.
- `<using class="XDHipaaRules">` selects a global rule class to be used.
This option also eliminates the need to specify the class on each individual `<rule>` element. In this case, the built-in HIPAA rule set, `XDHipaaRules`, is to be used.
You may also write your own custom rule set in a Java class and specify it here.
- `<rule tag="EL1">` indicates that a rule is to be applied to the segment or element called "EL1".

- `method="checkList"` identifies the actual rule to be applied.

This is a method of the global class being used as specified above, in this case "XDHipaaRules".

The checkList rule validates that an element contains only values from a defined list. There are many such built in rules (see the following Rule Table in [Reference: Syntax for Writing Rules](#)).

- `code="a,b,c"` is a parameter that the rule uses.

In this case, checkList would validate that the SEG1 element contains values from this list "a,b,c".

Each rule has a different set of parameters. (see the following Rule Table in [Reference: Syntax for Writing Rules](#)).

- `<rule tag="EL99" method="isFDate" code="RD8" />` is another rule; this one applied to an element named "EL99".

The rule to be applied is isFDate, which checks that the element value contains a date format.

In this case, the code attribute specifies which date format the value should be.

Reference: Syntax for Writing Rules

The following table lists the general syntax for writing rules.

Table 6-1 Rules Writing Syntax

Rule Element	Attribute	Description
<code><using></code>	<code>class=</code>	The Java program class containing all <code><rule></code> s within the section, unless overridden by a <code>class=</code> attribute in the <code><rule></code> entry itself.
<code><rule></code>	<code>tag=</code>	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, DTM causes 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, stag must be used.

Table 6-1 Rules Writing Syntax (Continued)

Rule Element	Attribute	Description
<rule>	stag=	For HIPAA documents, this is a specification subsection tag. This tag is explained in “Built-in HIPAA Rules” on page 5.
<rule>	name=	The rule’s identification, which should be a unique name. This is used in trace messages to specify which rule caused a violation. If omitted, no unique identification can be given.
<rule>	class=	The rule class to which this rule belongs. This corresponds to a Java object class, and each rule is a method of the class. If this is omitted, the class from the enclosing USING tag is used.
<rule>	method=	The specific rule.
<rule>	usage=	Specify usage=M (mandatory) to specify that there must be a value in the identified node. This check is applied before the actual rule logic is executed.

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

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.

Built-in HIPAA Rules

The Validation Engine provides a set of HIPAA rules (class=XDHipaaRules) that can be used to validate most HIPAA situations. These rules can be applied to any part of the incoming document. The rules make use of a standard set of attributes, as well as specialized attributes. Where the standard attributes are used, they are listed by name and not further described under each rule.

For HIPAA documents, the *tag=* attribute used to position the rule has been joined by a *stag=* (specification tag) attribute. One or the other may be used. *Stag=* positions to a specification section at the appropriate subchild. For example, *stag=BPR04* applies the rule to the fourth child of every BPR in the document.

Either *tag=* or *stag=* specification allows subsection specification by appending *:<subsection>* to the end of the tag. Subsections are base 1. For example, if the value of a field, ABC03 is HC:123:XY, to apply the isN rule to the 123 subsection, the address would be *stag=ABC03:2*. Regardless of the subfield separator character (specified in character 104 of the ISA segment), the colon is used in the addressing tag.

Condition Designator (cd=)

The standard X12 relational condition designators are supported in a list of designators. A fuller discussion of condition designators is found in *ASC X12N Insurance Subcommittee Implementation Guide, section A*. The rules engine accepts a list of standard designators separated by commas and/or blanks, for example:

```
<rule .. cd="R020305, C0403, P0506" ... >
```

There is no limit to the number of designators that can be specified. The designators are applied in order, and the first failure causes document rejection. Presence to the rule engine means that a child element of the parent exists and that it has a value. A child with no value is considered not present.

Reference: Supported Designators

Table 6-2 Supported Designators

Condition Code		Meaning	Definition
P	Paired or Multiple		If any element in the relational condition is present, then all of the specified elements must be present.
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.	
C	Conditional	If the first element in the condition list is present, then all other elements must be present. Elements not specified as the first element of the condition may appear without requiring that the first element be present. The element order in the condition is not critical beyond the first element.	
L	List Conditional		If the first element in the condition is present, then at least one of the remaining elements must be present. The element order in the condition is not critical beyond the first element.

If/Then Date Format Rules

Some segments contain a triplet consisting of a subfield:

1. A code.
2. A date or time format such as RD8.
3. A date or time value encoded as per the designated format.

The allowed format depends on the code. To accommodate this, the code= attribute can be an if/then set:

```
code="if/then, if/then..."
```

The if and then clauses allow several items, separated by a |. If the code is in the if list, then the format must be in the then list. If it is not in the if list, the rule steps to the next if list. For example:

```
code="416|19/D8|RD8, 22/TM, 77/"
```

To omit the then clause, use nothing after the '/'. This would signify that if the type is 77, in the previous example, then no date format or date is to be checked.

HIPAA Rule Set

Rules available for HIPAA validation include rules for single data items and rules for groups of documents. When encoding rules, use the highest-level rule possible. For example, use the DTM rules for a DTM rather than building the rules up from checkCD and checklist. Unless shown otherwise, rules are within XDHipaaRules which can be specified in a *using* tag.

checkDTM

checkDTM validates a generic DTM segment. The date is validated as per specification.

```
<rule tag="_835.DTM" method="checkDTM" code="102,307,582/N"/>
```

Table 6-3 checkDTM

Attribute	Meaning
cd	See “Condition Designator (cd=)” on page 6-5 .
code	<p>The value of the DTM01 element within the DTM must be one of the codes in the list. For example:</p> <pre><rule ... code="405,412" ...></pre> <p>The addition of a /N after a code means that the date field must be null. Otherwise the date must not be null.</p>

checkDTP

checkDTP validates a generic DTP segment. The date is validated as per specification.

```
<rule tag="DTP" method="checkDTP" code="405">
```

Table 6-4 checkDTP

Attribute	Meaning
cd	See “Condition Designator (cd=)” on page 6-5 .
code	<p>An if/then list for the 01/02/03 type/format/date-time subelements. See “If/Then Date Format Rules” on page 7 for format of an if/then list.</p>

checkDMG

checkDMG validates a generic DMG segment. The date is validated as per specification.

```
<rule tag="DMG" method="checkDMG" />
```

Table 6-5 checkDMG

Attribute	Meaning
cd	See “Condition Designator (cd=)” on page 6-5.

checkQTY

checkQTY validates a quantity of type and measure.

```
<rule tag="_2110.QTY" method="checkQTY"
code="NE,ZK,ZL,ZM,ZN,ZO" cd="R0204,E0204"/>
```

Table 6-6 checkQTY

Attribute	Meaning
cd	See “Condition Designator (cd=)” on page 6-5.
code	The value of the QTY01 element within the QTY must be one of the codes in the list.

checkCD

checkCD validates that a node has appropriate sub nodes.

```
<rule tag="NM1" method="checkCD" cd="P0809, C1110"/>
```

Table 6-7 checkCD

Attribute	Meaning
cd	See “Condition Designator (cd=)” on page 6-5 .

isN

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

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

Table 6-8 isN

Attribute	Meaning
min	Minimum number of digits required, not including sign. Optional.
max	Maximum number of digits permitted, not including sign. Optional.

isR

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

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

Table 6-9 isR

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.

isDate

isDate validates that a node is a CCYYMMDD format.

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

Table 6-10 isR

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

isTime

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

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

Table 6-11 isTime

Attribute	Meaning
None	Not applicable.

isFDATE

isFDate validates that a node has a proper date qualifier format such as RD8, based on the code list. If the qualifier is in the list, then the next field must be a date in the format as defined by the qualifier. If the date is null, the rule is successful.

```
<rule stag=CR603 method="isFDate" code="RD8" />
```

This example checks that the date value in field CR604 is formatted as per CR603.

Table 6-12 isFDATE

Attribute	Meaning
code	List of valid qualifiers.

checkLen

checkLen validates that a node is of sufficient size (length). Note that many rules provide minimum and maximum checking. In such cases, do not use this rule as it is redundant.

```
<rule stag=HI03:3 method="checkLen" min="2" max="8" />
```

Table 6-13 checkLen

Attribute	Meaning
min	Minimum number of characters. Optional. If omitted, no check is performed.
max	Maximum number of characters. Optional. If omitted, no check is performed.

checkUsage

checkUsage 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= or stag= attribute can be used to locate the section to which the rule applies. The domain= attribute specifies whether elements or components are to be tested. The following are three examples:

```
<rule name="simpler" tag="NAM" method="checkUsage" code="1=BK /
S2 + R3, 3 ! BR + 5=KK / N2 + S3"/>

<rule name="any" tag="NAM" method="checkUsage" code="1=?/ S2 +
R3"/>

<rule name="complex" tag="QQ" method="checkUsage" code="2:3=CD +
((1=XX | 3=BP) | 1=AB)/R1 + (N3:1 | R:2)" />
```

In the “simpler” example, the value of the NAM element is under examination. If the value in child 1 contains BK, then check the usages of components 2 and 3. If the value in component 3 is NOT BR, and the value in component 5 is KK, then component 2 must be null, and component 3 may be null.

In the “any” example, 1=?/xxx means that if field 1 has any code but is not empty, then the remainder of the rule is evaluated.

The “complex” example demonstrates that nested logical conditions are allowed on either the if or the then side of the equation.

The values to be checked are expressed as <child>:<part> where either is optional. If <child> addressing is used, 03 in SVC which is the third child of the SVC segment, then, while a *tag* could address SVC03, the *tag* should be used to address the SVC directly.

Note that + is used for “and” to avoid the need to escape the & entity.

Table 6-14 checkUsage

Attribute	Meaning						
Code	<p>The if/then validation criteria in the form</p> <pre>code := <item> [, <item>]* item := <if_exp> [+ <if_exp>]*/<then_spec> [+ <then_spec>]* if_exp := <position><op><value> then_spec := <action><position> position := child :composite child:composite child := integer composite := integer op := ! =</pre> <hr/> <p><action> codes</p> <hr/> <table> <tr> <td>R</td><td>A value in the field is required and must not be null.</td></tr> <tr> <td>N</td><td>The value in the field must be null, or the field must be missing.</td></tr> <tr> <td>S</td><td>The field may contain either a value or a null.</td></tr> </table> <hr/> <p>An if clause’s <value> of ? means that the then side applies regardless of the code value.</p>	R	A value in the field is required and must not be null.	N	The value in the field must be null, or the field must be missing.	S	The field may contain either a value or a null.
R	A value in the field is required and must not be null.						
N	The value in the field must be null, or the field must be missing.						
S	The field may contain either a value or a null.						

isCDate

isCDate validates that a node has a proper date qualifier format such as RD8, based on the code list. If the qualifier is in the list, then the next field must be a date in the format as defined by the qualifier. This differs from isFDate() in that it uses portions of the value field of the node for data, rather than following data fields.

```
<rule stag=CR603 method="isCDate" code="RD8" format="3",
value="4"/>
```

Table 6-15 isCDate

Attribute	Meaning
code	List of valid qualifiers or if/then list if <i>type</i> = used.
format	Subfield number (base 1) containing the format position to be checked.
value	Subfield number (base 1) containing the date value to be checked.
type	Optional. If used, the code must be an if/then format (see above) rather than a simple list. The <i>type</i> = attribute identifies the piece (base 1) containing the qualifier to test against the if side of the if/then rule.

checkList

checkList validates that the content of a field is in the list. This must address a single field. The list may be explicitly defined or in a supplied file.

```
<rule tag="NM1. _01_Entity_Identifier_Code_" method="checkList"
      code="BD,BS,FI,MC,PC,SL,UP,XX"/>
```

```
<rule tag="NM1. _01_Entity_Identifier_Code_" method="checkList"
      code="@ZIPCODES"/>
```

Table 6-16 checkList

Attribute	Meaning
code	One of the following: <ol style="list-style-type: none">1. A list of comma separated codes.2. The @ symbol to specify a file that contains the list. The name supplied is an alias that must be resolved in the Custom Dictionary <system><preload> section (see the example that follows this table).3. The name of a code list search routine. Code list search routines are Java classes that extend XDRuleList().

Example: Resolving a Checklist File Alias in the Custom Dictionary

This is the syntax required when using the checkList function with a file.

```
<system>
  <preload>
    <name
file="XDRuleListFile(C:\\HipaaCodes\\ZipCodes.txt)">ZIPCODES</name>
    </preload>
  </system>
```

The checkList supports a provided procedure named XDRuleListFile() that accepts a single parameter of the file name. The file must consist of a series of codes separated by blanks, commas or new lines. For example, to use a rule that employs one of these built-in code lists, enter the procedure into the dictionary using the console or add a <preload> entry to the <system> area of the dictionary. See the example in [“Writing Rule Search Routines in Java” on page 6-23](#).

checkEQ

checkEQ validates that if element a is present, element b must be present and equal to a. The elements a and b must be stags.

```
<rule tag="root" method="checkEQ" a="BPR10" b="TRB03"/>
```

Table 6-17 checkEQ

Attribute	Meaning
a	Value that triggers the rule.
b	Value that must be equal to a if a is present and has a value.

segXO

segXO exclusive or segment a or b may be present, but not both.

```
<rule tag="root" method="segXO" a="MIA" b="MOA"/>
```

Table 6-18 segXO

Attribute	Meaning
a	First value.
b	Second value.

compositeIntegrity

compositeIntegrity, given a tag addressed element of a segment such as HI01:1, then subsequent elements in the parallel elements such as HI02:1, must be from an if/then list.

```
<rule stag="HI01:1" method="compositeIntegrity" code="BR | XY/BR,
BP/AB|CD" />
```

Table 6-19 compositeIntegrity

Attribute	Meaning
code	Value of the independent variable sets the then list for subsequent elements.

relationalIntegrity

relationalIntegrity, given a tag addressed element of a segment such as NM101, then subsequent elements such as NM1xx, must be from an if/then list.

```
<rule stag="NM101" method="relationalIntegrity" code="BR | BK /
02.BR, BP/06.AB|CD, BP/03.XX" />
```

so that if 01 is BR, then NM102 must be BR. If 01 is BP, then NM106 must be AB or CD, and NM03 must be XX.

If the addressed sibling is missing or has no value, then the rule passes.

Table 6-20 relationalIntegrity

Attribute	Meaning
code	Value of the independent variable sets the then list for subsequent elements.

loopSegCount

loopSegCount, given a loop tag such as 1001A (which is the parent of 1001A.NM1s), then subsequent NM1 elements must be in a count list.

```
<rule tag="1001A" child="NM101" method="loopSegCount"
code="BR/1-1, BP/2-5 , CC/0-1" />
```

This means that if there is a BR in the NM101, then there must be 1 instance of NM101 with BR. If NM101 is BP, there can be at most 5 instances of NM101 with BP, and there must be at least 2 instances. If the code is CC, then there can be 1 instance, but it is not required. If the value is not in the list, ignore it.

Note that this rule addresses the segment loop, *not the data elements*.

Table 6-21 loopSegCount

Attribute	Meaning
code	Value of the independent variable sets the than list for subsequent elements. The format of an element of the code is <val>/<min>-<max>.

balance

balance balances a left term with a right term. The components must be children of the *tag* of the rule. Simple arithmetic is supported (plus, minus, one level of parenthesis).

```
<rule tag="_2110" method="balance" left="SVC03"
right="SVC02- (CAS03+CAS06+CAS09+CAS12+CAS15+CAS18) "/>
```

Table 6-22 balance

Attribute	Meaning
left	Value node for the left side of the equation.
right	Expression for the right side of the equation.

tranBal

tranBal is a specialized HIPAA 835 rule.

```
BPR02 = CLP04 - (PLB04+PLB06+PLB08+PLB10+PLB12)
<rule tag="root" class="XDHipaa835" method="tranBal"/>
```

Table 6-23 tranBal

Attribute	Meaning
No attributes	No parameters

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.

On entry to the rule, the following parameters are passed:

Table 6-24 Rules in Java

Parameter	Description
Node	The node identified by the tag attribute in the rule. The rule method is called once for each node that matches the tag specification.
Value	The 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	A <code>HashMap</code> of rule attributes. The rule method can check for any attributes that it desires. A <code>HashMap</code> is a fast implementation of a <code>Hashtable</code> that does not serialize.

Example: Writing Rules in Java

This section describes how to write rules in Java for special situations.

```
import java.util.*;
import com.ibi.edaqm.*;
public class XDMYRules extends XDRuleClass
{
    public XDMYRules()
    {
    }
    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 XDError(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:

Method	Purpose
Boolean isYYYYMMDD(String date)	Validates that a date is formatted correctly.
Boolean isInList(String list, String value)	The value must be in the list.
void trace(int level, String msg)	The text of the message is written to the system trace file. The level should be XD.TRACE_DEBUG, XD.TRACE_ERROR, or XD.TRACE_ALL.

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

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

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

Element	Meaning
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 AKx (997).
Explanation	A string explaining the error, for use when tracing .

Writing Rule Search Routines in Java

A short list can be searched by built-in rule engine code. A long list, where the values in the list are not obtained from the attribute directly but from an external source, require a rule list searcher tailored to the source. Lists can be obtained from a:

- Simple file
- Database with values loaded at startup
- Database with an access at each search request

Each list could 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 load at system initialization, and terminate at system shutdown. Each must offer a search method that determines whether the passed value is valid.

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

- `init(String[] args)` is called once at system initialization.
- `term()` is called once at system shutdown. It is not guaranteed to be called.
- `search(String value)` is called when the rule is executed.

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

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

This specifies that a rule can be written

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

that names the preloaded routine. This routine could load a list from a text file.

Example: Loading a Java File

The following is an example of loading a file containing codes:

```
import com.ibi.edagm.*;
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 XDEException
    {
        if (parms == null)
        {
            throw new XDEException(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);
}
}
```


7 Functional Acknowledgement Handling

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

- [Acknowledgement Processing](#)

Documents received by the BEA WebLogic Adapter for HIPAA are processed in stages that include preparse, validate, transform, acknowledgement, and routing. At any phase, the document may generate errors and may or may not pass specific validation rules. The validation engine and document validation rules are described in full in [Chapter 6, “Writing and Editing Rule Specification Files.”](#)

Acknowledgement Processing

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

- Validation

Validation is a specific stage in processing the document that occurs immediately after the document arrives and is available in XML format (that is, after the XML structure is available but before any other processing). The process of validation and the rules used in validating a document are described in [Chapter 6, “Writing and Editing Rule Specification Files.”](#)

- 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.

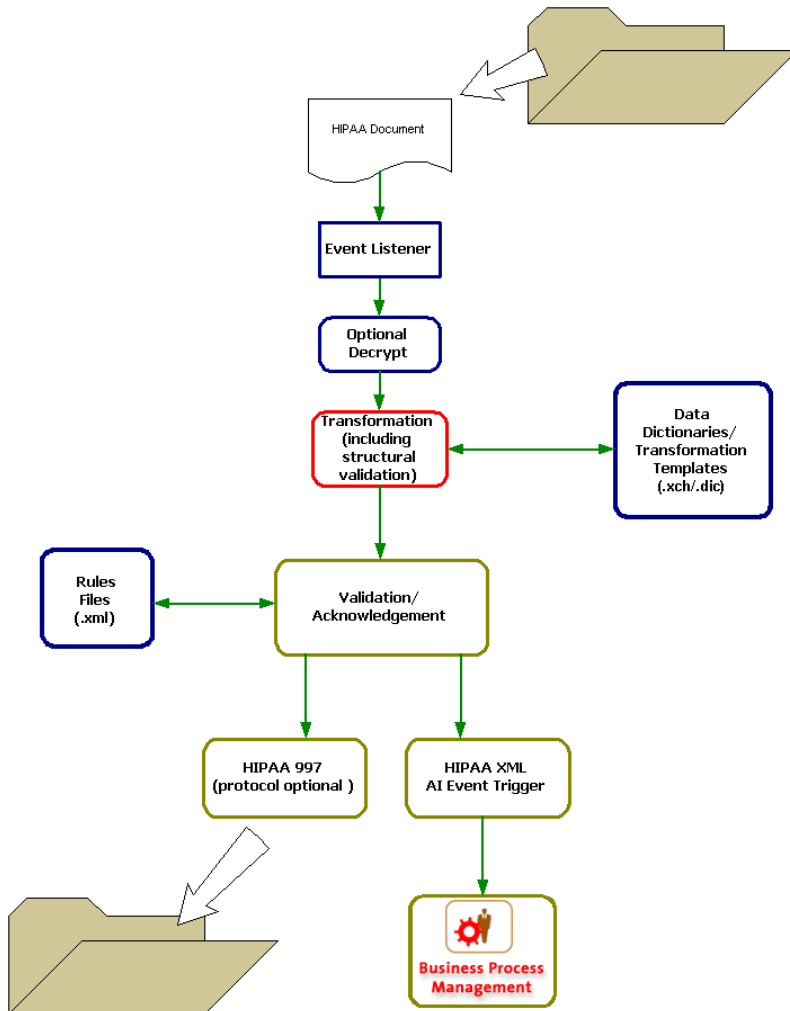
- Acknowledgement Agent

The Acknowledgement Agent is responsible for determining what processing is required for a document, as represented in memory by the Document Tree, and contains zero, one, or more validation errors. The agent creates the relevant functional acknowledgement (997 document) accordingly, based on the results of the validation process.

Documents, Validation, and Acknowledgement

Documents proceed from the Adapter Event Listener to the Event router and are processed in stages. This processing is shown in the following diagram:

Figure 7-1 Document Lifecycle



With respect to validation and acknowledgement, the previously illustrated document lifecycle has the following characteristics:

- Validation occurs as soon as the document has been converted into XML.
- Validation comprises both structural validation (described in dictionaries) and content and network validation (described in `Rules.xml` files).
- The validation processor (class) is defined at the document level or at the rule level. (See [Chapter 6, “Writing and Editing Rule Specification Files”](#) for a description.) An example validation processor is the `XDHIPAARules.class`.
- Validation processing adds Error elements into the Document Tree.
- The Acknowledgement Agent processes the document through its Document Tree, including any validation errors added during the previous validation phase.
- The output of the Acknowledgement Agent is independent of the output of the Document Agent (that is, different schema, separate thread of execution, and so forth).
- Validation errors and document output are independent of one another. In other words, a document may fail validation rules and have acknowledgements generated in the Acknowledgement Agent. However, the document is still passed to its agent and sent to its output unless specific actions are taken (that is, logic is coded).
- Acknowledgement processing can be customized to alter behavior when validation errors are present in the Document Tree.

Acknowledgement Agent

The Acknowledgement Agent is defined by the acknowledgement agent setting defined in the event JSP page. This is defaulted to the supplied acknowledgement agent provided with the product (HIPAA_997).

Figure 7-2 Edit Event

http://localhost:7001/BEA_HIPAA_1_0_Web/display.jsp - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address Links

Add Service
Add Event
Deploy Application View

Unique Event Name: * HIPAA_270

Select: File System

Location *	d:\filein\hipaa
File Suffix *	edi
Character Set Encoding *	ISO-8859-1
Polling Interval	1
Sort	<input type="checkbox"/>
Scan sub-directories	<input type="checkbox"/>
File-read limit (per scan)	
ackagent	HIPAA_997
protocol	
Path	
Transform Type *	270_to_XML

schema: 997

Acknowledgement Message Handling

The validation engine performs the content validation rules defined in the document specific `rules.xml` files. The Acknowledgement Agent generates an acknowledgement message based on the Document Tree. The message is a composite of the original XML document tree and any validation errors added by the validation engine. The results of the Acknowledgement Agent are dependent on the logic coded in the implementation class. For the BEA WebLogic Adapter for HIPAA, the default implementation class is the `XDHIPAAACKAgent.class` exposed as `HIPAA_997`.

The following is a sample output with errors:

Listing 7-1 Sample Output with Errors

```
<?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 [HIPAA835._835.DTM_]: code is
missing</error>
</eda>
```

The schema for the results of this acknowledgement is the `HIPAA_ACK.xsd` illustrated by the following:

Listing 7-2 Acknowledgement Result Schema

```
<?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="HIPAAack">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Error" minOccurs="0"
maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

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

Creating an Acknowledgement Event

In addition to the application view event created for the document, there must be an event created for the Acknowledgement Message generated by the Acknowledgement Agent. The following procedure creates an event for the acknowledgements generated by the `HIPAA_997` agent.

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

Figure 7-3 Adding a Functional Acknowledgement Event to the Event Adapter

http://localhost:7001/BEA_HIPAA_1_0_Web/display.jsp - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address Links

Add Service
Add Event
Deploy Application View

Unique Event Name: * HIPAA_270

Select: File System

Location *	d:\filein\hipaa
File Suffix *	edi
Character Set Encoding *	ISO-8859-1
Polling Interval	1
Sort	<input type="checkbox"/>
Scan sub-directories	<input type="checkbox"/>
File-read limit (per scan)	
ackagent	HIPAA_997
protocol	
Path	
Transform Type *	270_to_XML

schema: 997

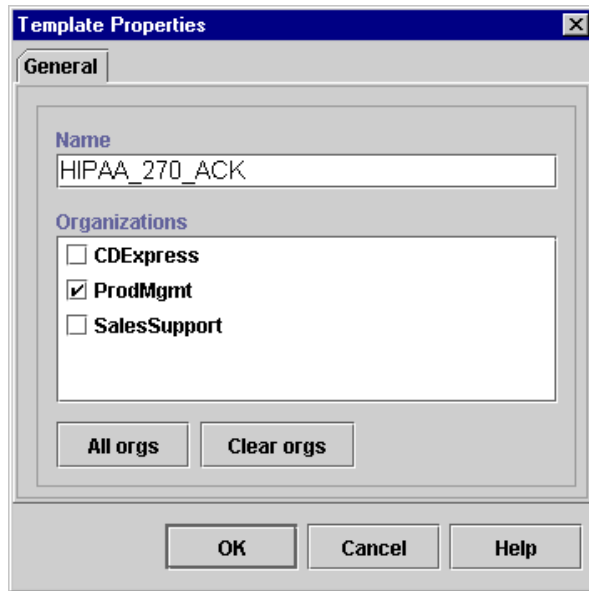
Note: The ackagent value is set to the desired Acknowledgement Agent (the default is set to HIPAA_997). Additionally, there is a 997 schema in the schema drop-down list.

It is important that the event adapter's protocol settings (in this case, MQSeries-based) are identical to those provided for the original event. If the settings are different, a separate Event Listener is created, and the two events (document and associated Functional Acknowledgement) are not tied together.

The Functional Acknowledgements created by the document are not seen by the event or schema combination created in this section. If you do not require the acknowledgement to be passed with the converted document to the workflow, you can opt to post the acknowledgement to be written to a file system. This is done by selecting the protocol option of FILE and setting the Path option to the path or filename mask, where the acknowledgement file is written. The protocol and path options must be omitted if passing the acknowledgement to the workflow.

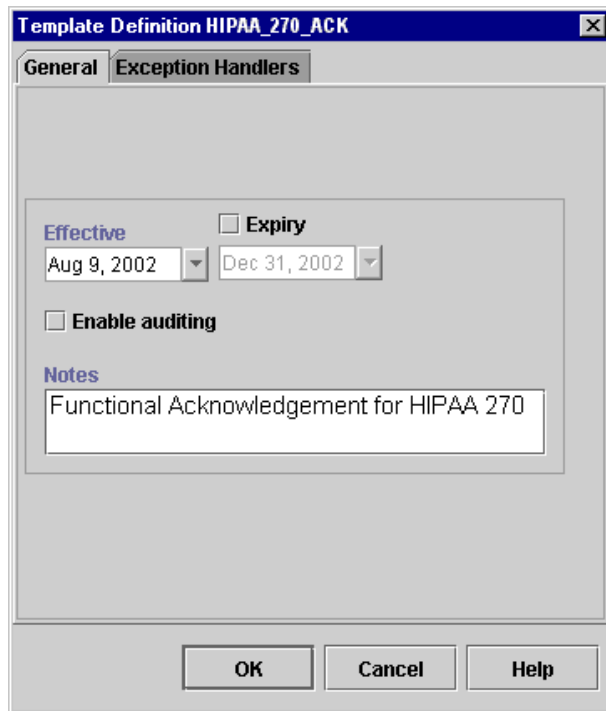
2. Add, continue, and deploy the application view.
3. From WebLogic Integration Studio, create a new WorkFlow Template.

Figure 7-4 Template Properties Dialog Box



4. When the Template Properties dialog box appears, enter a name for the template that indicates this workflow is for the Acknowledgement Message.
5. Click OK.
6. When the Template Definition dialog box opens, create a new Template Definition.
7. Click OK.

Figure 7-5 Template Definition Dialog Box



8. Open the new Template Definition, select the Start object, and complete the properties.

Figure 7-6 Event Definition Start Properties

Start Properties

Description

☐ Timed
 ☐ Manual
 ☐ Called
 ☒ Event
 AI Start

Root

- MQSeries
- MQTesting
- SAP
- DesignScenarios
- HIPAA EDI
 - CLAIMS
 - Coverage Confirmation
 - 270 Eligibility request
 - 270_Ack

Name: 270_ACK

Description:

Condition:

Event Document Variable:

<new>

Start Organization

ProdMgmt

☐ Use workflow expression

Variables Actions Next Notes

Variable	Expression
----------	------------

Add

Update

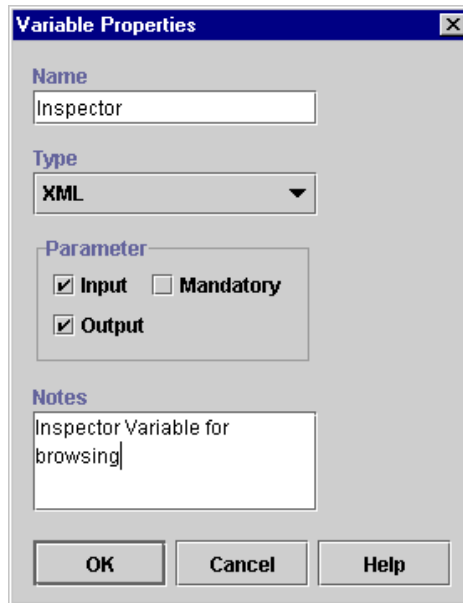
Delete

OK Cancel Help

- a. Select Start →AI Event.

- b. Select HIPAA→HIPAASecurityEvent→Functional Acknowledgment event in the left AI event pane.
- c. Choose the Start Organization to be the same as the Template Definition Organization.
- d. Add a new Event Document Variable.

Figure 7-7 Add Start Variable

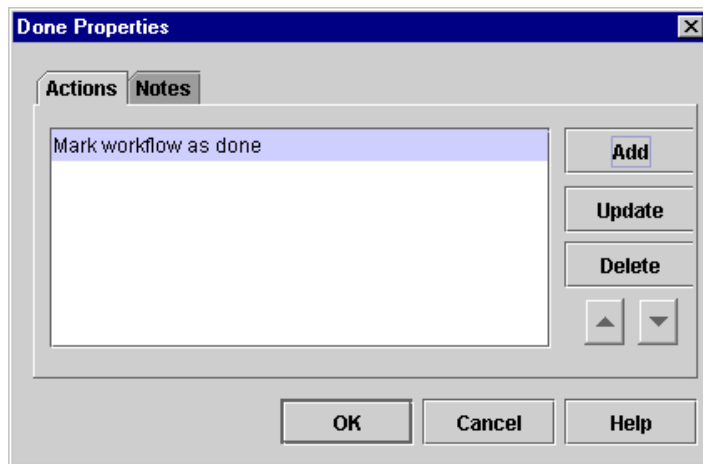


The image shows a 'Variable Properties' dialog box with a blue title bar and a close button. It contains several sections: 'Name' with a text field containing 'Inspector'; 'Type' with a dropdown menu set to 'XML'; 'Parameter' with two checked checkboxes, 'Input' and 'Output', and an unchecked 'Mandatory' checkbox; and 'Notes' with a text area containing 'Inspector Variable for browsing'. At the bottom are three buttons: 'OK', 'Cancel', and 'Help'.

Variable Properties	
Name	
Inspector	
Type	
XML	
Parameter	
<input checked="" type="checkbox"/> Input	<input type="checkbox"/> Mandatory
<input checked="" type="checkbox"/> Output	
Notes	
Inspector Variable for browsing	
OK	Cancel Help

- e. Choose Input and Output parameter type.
9. Add an Action to the Done object.
 - a. Choose the Done object.
 - b. Click Add an Action.
 - c. Select Task Actions and choose Mark Workflow as Done.

Figure 7-8 Done Properties Dialog Box - Mark Workflow as done



10. Click OK in the Done Properties dialog box.
11. Right-click Message Definition in the left pane and select Save.
12. Ensure the workflow is active by selecting the Properties of the Workflow Definition.

Figure 7-9 Template Definition Properties

Template Definition HIPAA_270

General Exception Handlers

Workflow Label

☒ Active

Effective ☐ Expiry

Aug 9, 2002 Dec 31, 2002

☐ Enable auditing

Notes

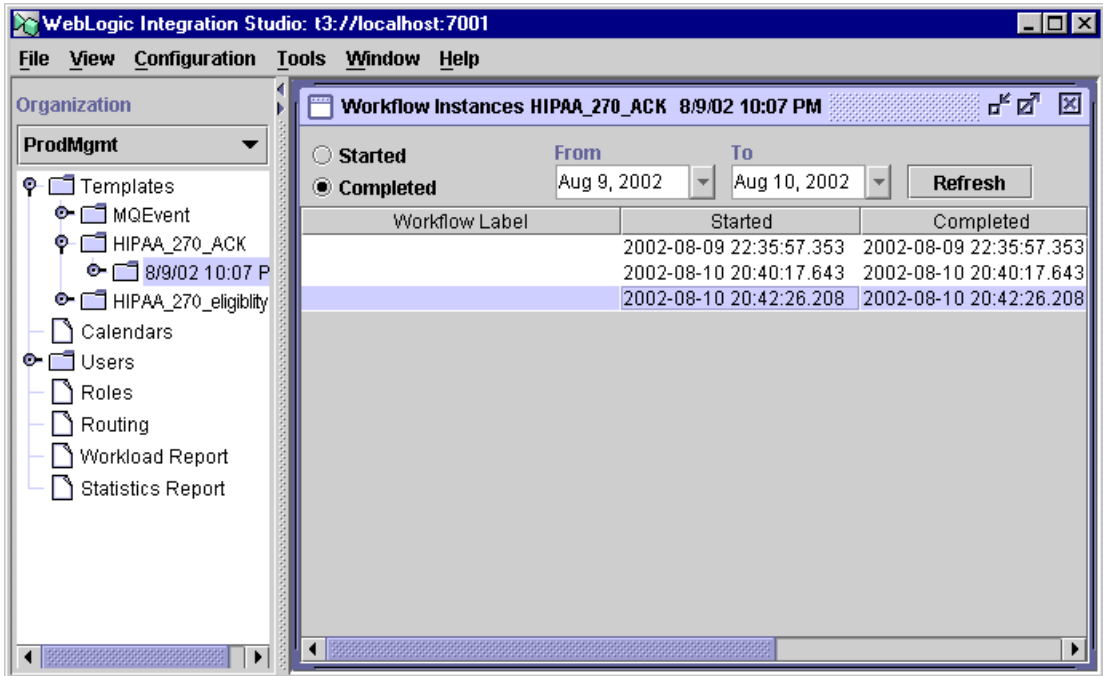
OK Cancel Help

Testing Acknowledgement Message Handling

Having created a HIPAA event adapter with two registered events, a HIPAA message event (for example, HIPAA_270) and a HIPAA Functional Acknowledgment message event, you may view the document as it has been processed through the workflow in the WebLogic Studio console.

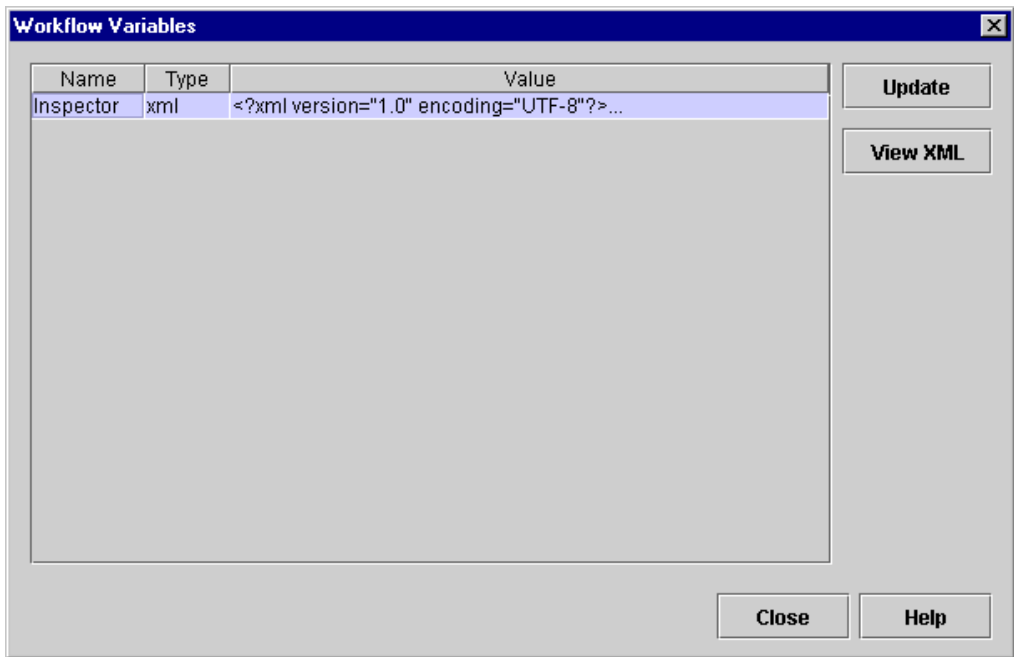
1. Right-click Templates and select Instances.

Figure 7-10 Template Workflow Instances



2. Choose the instance of interest (that is, the instance generated by the bad message).
3. Right-click and select Workflow Variables:

Figure 7-11 Workflow Variables Dialog Box



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

Figure 7-12 Workflow XML Variable

