BEA WebLogic Adapter for HIPAA®

User Guide

Version 8.1.2
Document Revised: January 2004

Copyright

Contents

About This Document

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

- Chapter 1, "Introducing the BEA WebLogic Adapter for HIPAA," describes the adapter, how it relates to both HIPAA business objects and WebLogic Integration.
- Chapter 2, "Generating Schemas for HIPAA Documents," describes how to generate schemas for your HIPAA business objects.
- Chapter 3, "Defining Application Views for HIPAA," describes application views and how to use them to configure events and services.
- Chapter 4, "Writing and Editing Rule Specification Files," describes how to write your own rule files.
- Chapter 5, "Functional Acknowledgement Handling," describes how to use the acknowledgement agent.

Who Should Read This Documentation

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

- Integration Specialists—Lead the integration design effort. Integration specialists have expertise in defining the business and technical requirements of integration projects, and in designing integration solutions that implement specific features of WebLogic Integration. The skills of integration specialists include business and technical analysis, architecture design, project management, and WebLogic Integration product knowledge.

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

Additional Information

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

- *BEA WebLogic Adapter for HIPAA Release Notes*

  `http://edocs.bea.com/wladapters/hipaa/docs812/pdf/relnotes.pdf`
- *BEA WebLogic Adapter for HIPAA Installation and Configuration Guide*

  `http://edocs.bea.com/wladapters/hipaa/docs812/pdf/install.pdf`
- *Introduction to the BEA WebLogic Adapters*

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

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

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

  `http://edocs.bea.com/wli/docs81/aiuser/index.html`
- BEA WebLogic Integration documentation

  `http://edocs.bea.com/wli/docs81/index.html`
- BEA WebLogic Platform documentation

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

- HIPAA documentation

  http://www.hipaa.org

How to Use This Document

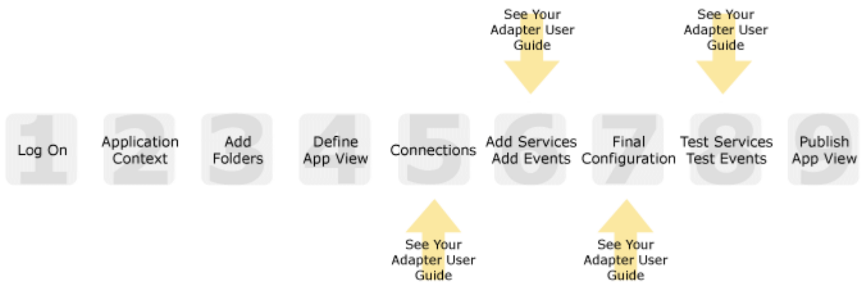
This document is designed to be used in conjunction with *Using the Application Integration Design Console*, available at the following URL:

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

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

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

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


See Your Adapter User Guide
See Your Adapter User Guide
1 Log On
2 Application Context
3 Add Folders
4 Define App View
5 Connections
6 Add Services Add Events
7 Final Configuration
8 Test Services Test Events
9 Publish App View
See Your Adapter User Guide
See Your Adapter User Guide

Contact Us!

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

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

If you have any questions about this version of BEA WebLogic Integration Adapter for HIPAA, or if you have problems using the BEA WebLogic Integration 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
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

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

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

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

Convention	Item

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

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

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

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

CHAPTER 1

Introducing the BEA WebLogic Adapter for HIPAA

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

It includes the following topics:

- About the BEA WebLogic Adapter for HIPAA
- Getting Started With the Adapter for HIPAA

About the BEA WebLogic Adapter for HIPAA

HIPAA mandates the use of standards whenever healthcare entities send automated transactions to one another. These standards support Electronic Data Interchange (EDI) of administrative and financial health care transactions between health care providers and plans.

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

This section includes the following topics:

- Supported HIPAA Operations for Application Integration
- Supported Services
- Supported Events
- Benefits of the Adapter for HIPAA

Supported HIPAA Operations for Application Integration

The Adapter for HIPAA supports synchronous and asynchronous, bi-directional message interactions for systems using HIPAA documents.

It provides integration with the following HIPAA operations:

- Access to your HIPAA system using XML to handle both services and events
- Generation of functional acknowledgements, also known as 997 documents
- Support for the HIPAA standard documents

Note: The adapter does not provide out-of-the-box support for customizing the standard HIPAA documents. Please contact BEA professional services if you need to customize these formats.

Supported Services

The Adapter for HIPAA supports four types of services, one for each type of HIPAA transports: File, FTP, HTTP, MQ, and TCP. In each case, the adapter sends a file to your HIPAA system.

These are the services supported by Adapter for HIPAA:

- File service, which sends a file to a specific directory on disk.
- FTP service, which sends a file through FTP.
- HTTP service, which sends a file through HTTP.
- MQ service, which sends a file to an IBM MQSeries or WebSphere MQ queue.
- TCP service, which sends a file through TCP.

Supported Events

The Adapter for HIPAA supports four types of services, one for each type of HIPAA transports: File, FTP, HTTP, MQSeries, and TCP. In each case, the adapter picks up a file and passes it to your business process.

These are the events supported by Adapter for HIPAA:

- File event, in which the adapter picks up a file from a specific directory on disk.
- FTP event, in which the adapter receives a file through FTP.

- HTTP event, in which the adapter receives a file through HTTP
- MQ event, in which the adapter picks up a file from a specific IBM MQSeries or WebSphere MQ queue.
- TCP event, in which the adapter receives a file through TCP.

Benefits of the Adapter for HIPAA

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

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

  `http://java.sun.com/j2ee/connector/`
- The adapter and WebLogic Integration solution is scalable. The BEA WebLogic Platform provides clustering, load balancing, and resource pooling for a scalable solution. For more information about scalability, see the following URL:

  `http://edocs.bea.com/wls/docs81/cluster/index.html`
- The adapter and WebLogic Integration solution benefits from the fault-tolerant features of the BEA WebLogic Platform. For more information about high availability, see the following URL:

  `http://edocs.bea.com/wli/docs81/deploy/index.html`
- The adapter and WebLogic Integration solution is secure, using the security features of the BEA WebLogic Platform and the security of your HIPAA system. For more information about security, see the following URL:

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

Getting Started With the Adapter for HIPAA

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

- Step 1: Design the Application Integration Solution
- Step 2: Determine the Required HIPAA Business Workflows
- Step 3: Generate Schemas for HIPAA Documents
- Step 4: Define Application Views and Configure Services and Events
- Step 5: Integrate Your HIPAA System with Other BEA Software Components
- Step 6: Deploy the Solution to the Production Environment

Step 1: Design the Application Integration Solution

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

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

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

Step 2: Determine the Required HIPAA Business Workflows

Within the larger context of an application integration project, you must determine which specific HIPAA documents are required for services and events to support the business processes in the application integration solution.

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

- Type of HIPAA documents and transport used to access the HIPAA system
- HIPAA transactions involved in business processes
- Logins required to access HIPAA transports and perform the required operations
- Whether operations are, from the adapter point of view:
  - services, which notify the HIPAA system with a request for action, and, in addition, whether such services should be processed synchronously or asynchronously
  - events, which are notifications from the HIPAA system that trigger workflows

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

Step 3: Generate Schemas for HIPAA Documents

After identifying the HIPAA documents required for the application integration solution, you must generate the XML schemas that will be used to exchange data with one or more HIPAA systems:

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

To learn more about schemas, see Chapter 2, "Generating Schemas for HIPAA Documents."

Step 4: Define Application Views and Configure Services and Events

After you create the schemas for your HIPAA services or events, you create an application view that provides an XML-based interface between WebLogic Server and a particular HIPAA system. If you are accessing multiple HIPAA systems, you define a separate application view for each HIPAA system you want to access. To provide different levels of security access (such as "guest" and "administrator"), define a separate application view for each security level.

Once you define an application view, you can configure events and services in that application view that employ the XML schemas that you created in "Step 3: Generate Schemas for HIPAA Documents" on page 1-5. To learn more about generating schemas, see Chapter 2, "Generating Schemas for HIPAA Documents."

To learn more about defining application views, see Chapter 3, "Defining Application Views for HIPAA" in conjunction with *Using the Application Integration Design Console*, at the following URL:

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

Step 5: Integrate Your HIPAA System with Other BEA Software Components

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

For more information, see *Using the Application Integration Design Console*, particularly Chapter 3, "Using Application Views with Application Workflows," at the following URL:

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

Step 6: Deploy the Solution to the Production Environment

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

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

Generating Schemas for HIPAA Documents

The Adapter for HIPAA uses XML documents to communicate with your HIPAA system for both services and events. The format of these XML documents is determined by schemas that are generated automatically for you by the adapter. This section contains the following topics:

- About Schemas
- About Validating HIPAA Messages
- About Schema Repositories
- Next Steps

About Schemas

Each service or event the Adapter for HIPAA uses is defined by a schema. All of the documents the adapter sends to, or receives from the EIS must be defined by schemas. These schemas are used as part of the message validation process. To learn more about message validation, see "About Validating HIPAA Messages" on page 2-2. The adapter uses the following schemas:

- Service Requests
- Service Responses
- Events

Service Requests

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

Service Responses

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

Events

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

About Validating HIPAA Messages

HIPAA messages are validated in two ways. First, the structure of the message is checked to make sure it conforms to the schema for its message type. Second, the content of the file is validated by the rules engine. The rules engine uses a rules file that contains pre-configured rules for the elements of the HIPAA message. You can customize these rules, and you can also write your own rules to apply your own business logic.

About Rules Files

After the document has been converted from the HIPAA EDI format into XML, using the schema, the adapter uses the rules files to validate the contents of the document. The rules files are pre-built to apply the HIPAA-mandated rules. Rules are associated with the converted document by the document's root tag. The BEA WebLogic Adapter for HIPAA is pre-configured to apply these rules to the document. To learn more about these documents, see Chapter 4, "Writing and Editing Rule Specification Files."

About EDI to XML Transformation

The BEA WebLogic Adapter for HIPAA automatically transforms HIPAA EDI documents to XML and vice versa. In addition, you can use a business process workflow to build templates of business processes to convert to application-specific XML or EDI form.

About Schema Repositories

The schema repository stores schema information. This section contians the following topics:

- About the Contents of the Schema Repository
- About the Repository Manifest

About the Contents of the Schema Repository

The adapter automatically generates repository directories and components.

A schema repository consists of the following elements:

- Manifest file (`manifest.xml`) that describes the event and service schemas contained in the repository.
- Event and service schemas. The schemas are usually stored in files with an `.xsd` extension.
- HIPAA document dictionaries.

  HIPAA documents are described by dictionary files. Dictionaries are used for validation.
- Rules files. These rules files initiate the validation process by applying rules to the data. To learn more about rules files, see Chapter 4, "Writing and Editing Rule Specification Files."
- Code sets. These are files that are used to do some checking of the data values.

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

About the Repository Manifest

Each schema repository has a manifest that describes the repository and its contents. The repository manifest is an XML file named `manifest.xml`. This file is created automatically when the repository is generated. The root directory for the repository is based on your version of HIPAA (version 2.3, 2.31, or 2.4).

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

Listing 2-1 Sample Repository Manifest (Portion Only)

```
</manifest>
   <connection/>
   <schemaref name="270">
      <request root="HIPAA270" file="HIPAA270.xsd"/>
      <response root="emitStatus" file="FileEmit.xsd"/>
      <event root="HIPAA270" file="HIPAA270.xsd"/>
   </schemaref>
</manifest>
```

The repository has a connection section, which can be ignored for this adapter. It also has a schema reference section, named `schemaref`. The schema reference name appears in the drop-down list on the Add Service or Add Event screens in the WebLogic Integration Application View Console. Each named schema reference can contain three schemas, one of each type.

Next Steps

The next step is to create an application view. An application view makes the services and events available to applications. To learn more about application views, see Chapter 3, “Defining Application Views for HIPAA.”.

Defining Application Views for HIPAA

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

This section presents the following topics:

How to Use This Document

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

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

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

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

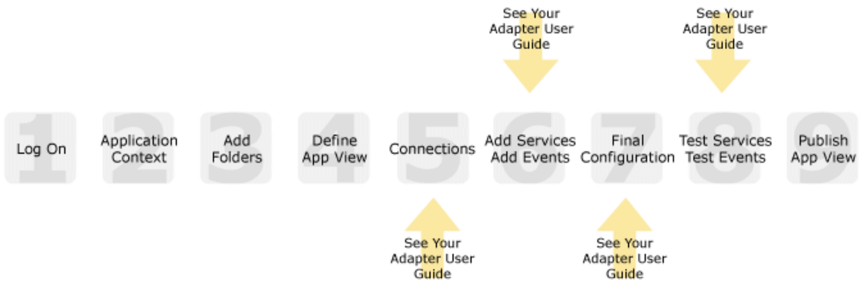

See Your Adapter User Guide
See Your Adapter User Guide
1 Log On
2 Application Context
3 Add Folders
4 Define App View
5 Connections
6 Add Services Add Events
7 Final Configuration
8 Test Services Test Events
9 Publish App View
See Your Adapter User Guide
See Your Adapter User Guide

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

Before You Begin

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

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

processes, you can define and test the appropriate services and events. For more information, see "Getting Started With the Adapter for HIPAA" on page 1-4.

- Gathered the connection information for your HIPAA system. To learn more about the connection information needed for your HIPAA system, contact your HIPAA system administrator.

About Application Views

An application view defines:

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

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

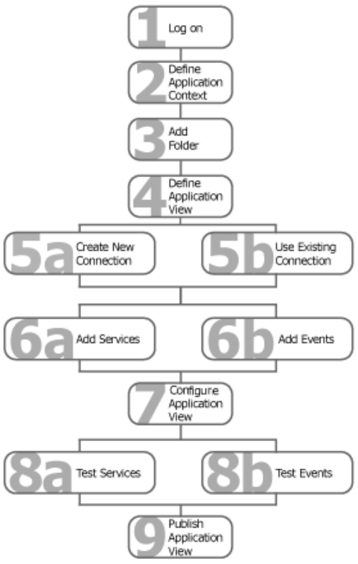
About Defining Application Views

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

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

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

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


1 Log on
2 Define Application Context
3 Add Folder
4 Define Application View
5a Create New Connection
5b Use Existing Connection
6a Add Services
6b Add Events
7 Configure Application View
8a Test Services
8b Test Events
9 Publish Application View

To define an application view:

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

   This application will be using the events and services you define in your application view. The application view works within the context of this application.
3. Add folders as required to help you organize application views.
4. Define a new application view for your adapter.
5. Add a new connection service or select an existing one.

   If you are adding a new connection service, see "Defining Service Connection Parameters" on page 3-5 for details about HIPAA system requirements.
6. Add the events and services for this application view.

   See the following sections for details about HIPAA system requirements:

   – "Setting Service Properties" on page 3-6

   – "Setting Event Properties" on page 3-15

7. Perform final configuration tasks.

   If you are adding an event connection, see "Defining Event Connection Parameters" on page 3-27 for details about HIPAA system requirements.

8. Test all services and events to make sure they can properly interact with the target HIPAA system.

   See the following sections for details about HIPAA system requirements:

   – "Testing Services" on page 3-29

   – "Testing Events Using a Service" on page 3-30

   – "Testing Events Manually" on page 3-31

9. Publish the application view to the target WebLogic Workshop application.

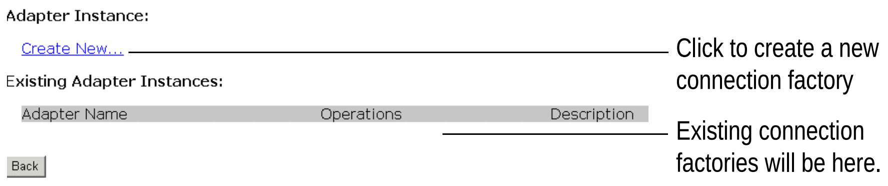
   This is the application you specified in step 2. Publishing the application view allows workflow developers within the target application to interact with the newly published application view using an Application View control.

Defining Service Connection Parameters

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

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

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

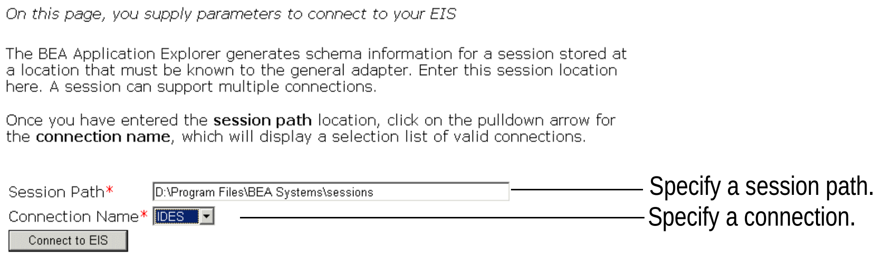

Adapter Instance:
Create New...
Click to create a new connection factory
Existing Adapter Instances:
Adapter Name
Operations
Description
Existing connection factories will be here.
Back

After you enter a connection name and description, you use the Configure Connection Parameters page to specify connection parameters for a connection factory.

To create a new browsing connection:

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

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



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

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

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

Session Path* D:\Program Files\BEA Systems\sessions

Connection Name* IDES

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

2. Specify a session path and connection name.

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

3. Click Connect to EIS.

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

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

Setting Service Properties

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

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

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

- File Service

- FTP Service
- HTTP Service
- MQ Service
- TCP Service

File Service

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

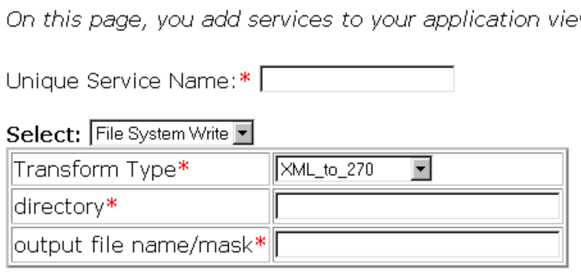
http://edocs.bea.com/wli/docs81/aiuser/index.html

A File service sends a file to a specific directory on disk. The HIPAA system responds to the service request it receives from the adapter and responds with a service response.

To configure a File Service:

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

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

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

   Unique Service Name:*

   Select: File System Write

Transform Type*	XML_to_270
directory*	
output file name/mask*	

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

3. Enter the following information:

Table 3-1 File Service Parameters

Parameter	Description
Transform Type	Select the pre-built transform template from the drop-down box. All template names are prefixed with XML_to *documentType*.
directory	The target file system location for the file
output file name/mask	The file name to be used for the output file generated as a result of this operation

4. See "Common Service and Event Settings" on page 3-14 for information about selecting a schema and configuring tracing.

FTP Service

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

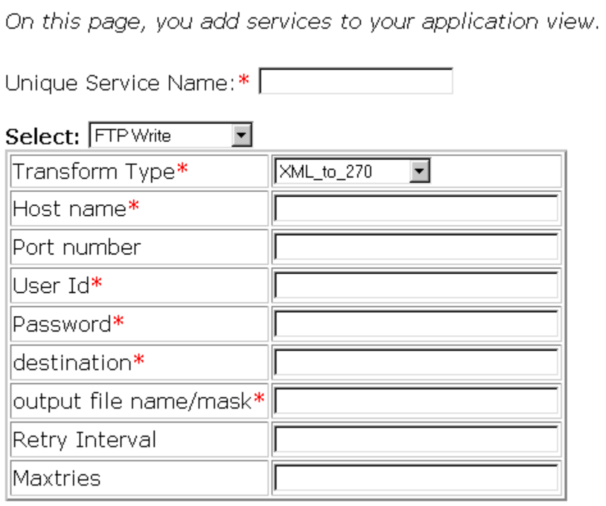
`http://edocs.bea.com/wli/docs81/aiuser/index.html`

An FTP service sends a file through HTTP to the HIPAA system. The HIPAA system responds to the file it receives from the adapter with a service response.

To configure an FTP Service:

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

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



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

Unique Service Name:*	
Select:	FTP Write
Transform Type*	XML_to_270
Host name*	
Port number	
User Id*	
Password*	
destination*	
output file name/mask*	
Retry Interval	
Maxtries	

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

3. Enter the following information:

Table 3-2 FTP Service Parameters

Parameter	Description
Transform Type	Select the pre-built transform template from the drop-down box. All template names are prefixed with XML_to *documentType*.
Host name	The name of the FTP host
Port number	Port number to use for the FTP connection. Leave this field blank to use the default port number.
User ID	The user name to log in to the FTP server
Password	The password for the FTP user
destination	The directory on the FTP server to which to write the file

Table 3-2 FTP Service Parameters (Continued)

Parameter	Description
output file name/mask	The file name to be used for the output file generated as a result of this operation. A * in the filename expands to a timestamp. A # in the filename is a mask for a sequence count. Each pound symbol (#) represents an 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.
Retry Interval	The maximum wait interval betewwn retires when a connection fails. This is in the format xxH:xxM:xxS For example, 1 hour, 2 minutes and 3 seconds is: 1H:2M:3S
Maxtries	The maximum number of retires to attempt when the adapter fails to connect or write the file

4. See “Common Service and Event Settings” on page 3-14 for information about selecting a schema and configuring tracing.

HTTP Service

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

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

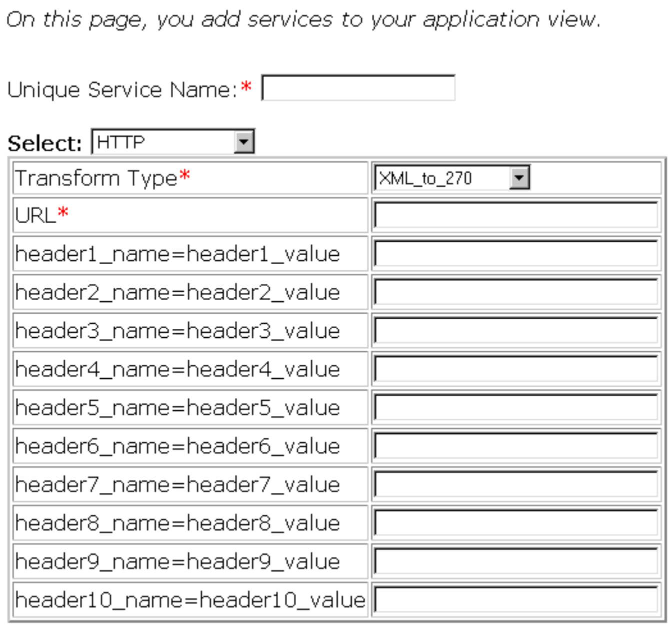
`http://edocs.bea.com/wli/docs81/aiuser/index.html`

An HTTP service sends a file through HTTP to the HIPAA system. The HIPAA system responds to the file it receives, sending a service response to the adapter.

To configure an HTTP Service:

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

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


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

Unique Service Name:*

Select: HTTP

Transform Type*	XML_to_270
URL*	
header1_name=header1_value	
header2_name=header2_value	
header3_name=header3_value	
header4_name=header4_value	
header5_name=header5_value	
header6_name=header6_value	
header7_name=header7_value	
header8_name=header8_value	
header9_name=header9_value	
header10_name=header10_value	

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

3. Enter the following information:

Table 3-3 HTTP Service Parameters

Parameter	Description
Transform Type	Select the pre-built transform template from the drop-down box. All template names are prefixed with XML_to *documentType*.
URL	The HTTP-compliant URL where the adapter is to post the file
header_name=header_value	Ten optional headers and values that can be passed in the post operation. Use the following format: headername=headervalue

4. See "Common Service and Event Settings" on page 3-14 for information about selecting a schema and configuring tracing.

MQ Service

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

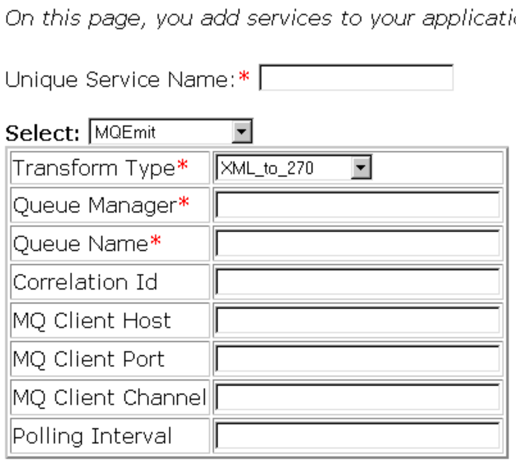
http://edocs.bea.com/wli/docs81/aiuser/index.html

An MQ service sends a file to an IBM MQSeries or WebSphere MQ queue. The HIPAA system responds to the file it receives from the adapter and returns a service response.

To configure an MQ Service:

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

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



On this page, you add services to your application

Unique Service Name:*

Select: MQEmit

Transform Type*	XML_to_270
Queue Manager*	
Queue Name*	
Correlation Id	
MQ Client Host	
MQ Client Port	
MQ Client Channel	
Polling Interval	

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

3. Enter the following information:

Table 3-4 MQ Service Parameters

Parameter	Description
Transform Type	Select the pre-built transform template from the drop-down box. All template names are prefixed with XML_to *documentType*.

Table 3-4 MQ Service Parameters (Continued)

Parameter	Description
Queue Manager	The name of the MQ Queue Manager to be used
Queue Name	The name of the MQSeries or WebSphere MQ queue that the HIPAA system polls
Correlation Id	The correlation ID used in the MQ message header
MQ Client Host	For MQ Client only. The host where the MQ Server is located
MQ Client Port	For MQ Client only. The port number used to connect to an MQ Server
MQ Client Channel	For MQ Client only. The channel between an MQ Client and MQ Server
Polling Interval	This is a time, expressed as xxH:xxM:xxS For example 1 hour, 2 minutes, and 3 seconds is: 1H:2M:3S The maximum 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 a worker thread cannot respond to a stop command. If this value is set to 0, the listener runs once and terminates. The default value is 2 seconds.

4. See "Common Service and Event Settings" on page 3-14 for information about selecting a schema and configuring tracing.

TCP Service

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

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

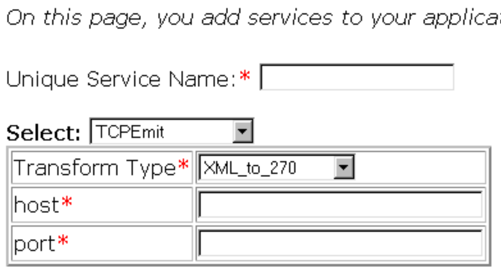
A TCP service sends a file to the HIPAA system via TCP. The HIPAA system responds to the file it receives from the adapter and returns a service response.

To configure a TCP Service:

1. Enter a unique service name that describes the function the service performs.

2. Select TCPEmit from the Select list.

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


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

Unique Service Name:*	
Select:	TCPEmit
Transform Type*	XML_to_270
host*	
port*	

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

3. Enter the following information:

Table 3-5 TCP Service Parameters

Parameter	Description
Transform Type	Select the pre-built transform template from the drop-down box. All template names are prefixed with XML_to *documentType*.
host	The name or address of the client restricted to accessing this adapter
port	The TCP port the adapter listens on

4. See "Common Service and Event Settings" on page 3-14 for information about selecting a schema and configuring tracing.

Common Service and Event Settings

1 2 3 4 5 6 7 8 9

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

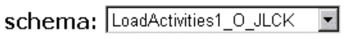
`http://edocs.bea.com/wli/docs81/aiuser/index.html`

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

To set common service settings:

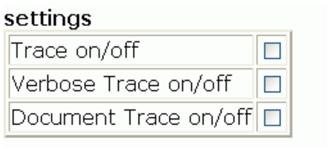
1. In the Schema list, select the schema you want to use with this service.

   For more information, see Chapter 2, "Generating Schemas for HIPAA Documents."


schema: LoadActivities1_O_JLCK

2. Configure logging and tracing for this service, as follows:

   Logging captures information from your adapter and writes it in a log file. Tracing displays runtime information in the console. You set the type and amount of information you wish to capture as part of the final configuration tasks. This is described in detail in *Using the Application Integration Design Console*.

   
   settings

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

   a. Select the Trace on/off check box to enable tracing for this service. Trace information appears in the runtime console.

   b. Select the Verbose Trace on/off check box to enable additional trace information for deeper troubleshooting.

   c. Select the Document Trace on/off check box to enable recording of the documents sent and received by the adapter in the trace information for this service.

3. Click Add to add the service.

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

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

Setting Event Properties

1 2 3 4 5 6 7 8 9

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

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

An event defines how your application responds to events generated by HIPAA. The Adapter for HIPAA supports the following events:

- File Event
- FTP Event

- HTTP Event
- MQ Event
- TCP Event

File Event

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

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

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

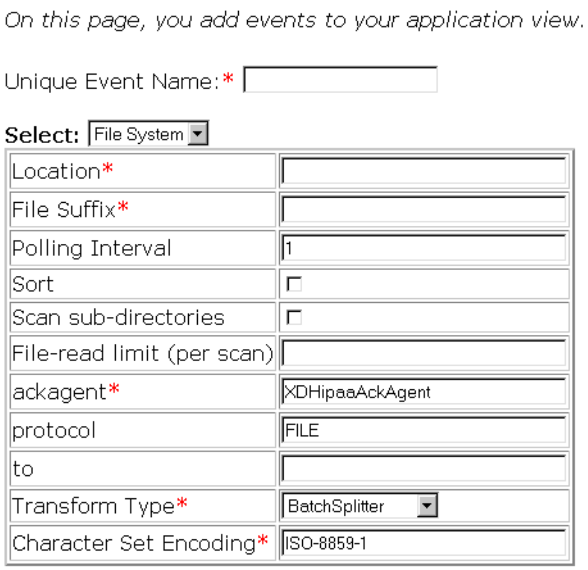
In a File Event, the adapter picks up a file from a specific location on disk and passes it to your business process.

When you create an event for your HIPAA application view using the File System protocol, batch processing is now available. To learn more about batch processing, see Using Batch Processing.

To configure a File Event:

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

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



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

Unique Event Name:*

Select: File System

Location*	
File Suffix*	
Polling Interval	1
Sort	☐
Scan sub-directories	☐
File-read limit (per scan)	
ackagent*	XDHipaaAckAgent
protocol	FILE
to	
Transform Type*	BatchSplitter
Character Set Encoding*	ISO-8859-1

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

3. Enter the following information:

Table 3-6 File Event Parameters

Parameter	Description
Location	The directory where messages are received. DOS-style file patterns are valid for this parameter. You can specify a file pattern as well as a directory. For example, `c:\xyz\ab*cd` (without a file suffix) takes the file suffix from that parameter. If you use a pattern, files are selected based on the suffix and then the pattern. AB?CD selects ABxCD. AB*CD selects ABxxxCD.
File Suffix	File extension for the file event. This limits input files to those with the specified extensions. The "." is not required. The minus sign ("-") indicated that there is no extension. If the file extension is `zip`, the unzipped files must conform to the event schema, or they will fail. This function also works with transform configured.

Table 3-6 File Event Parameters (Continued)

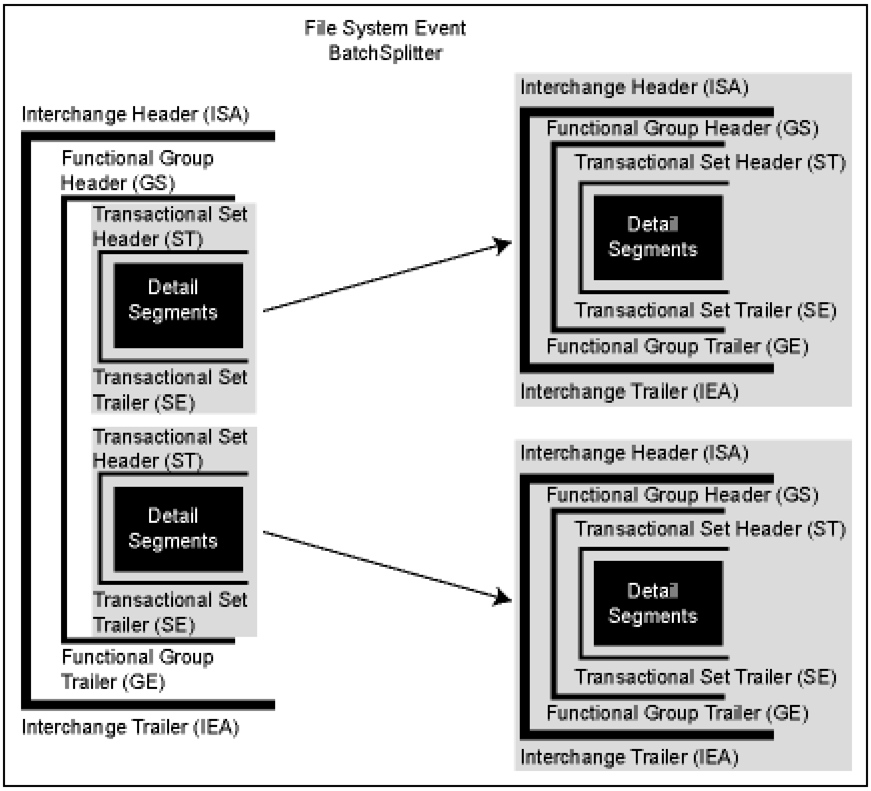
Parameter	Description
Polling Interval	This is a time, expressed as xxH:xxM:xxS For example 1 hour, 2 minutes, and 3 seconds is: 1H:2M:3S The maximum interval between checks for new documents. The higher this value, the longer the interval, and the fewer system resources that are used. The side-effect of a high value is that a worker thread cannot respond to a stop command. If this value is set to 0, the listener runs once and terminates. The default value is 2 seconds.
Sort	Sort incoming documents by arrival time. This is a boolean, valid values are TRUE or FALSE. Setting this value to TRUE maintains the sequence of incoming documents, but may slow performance.
Scan sub-directories	Setting for scanning subdirectories for document to be processed. This is a boolean, valid values are TRUE or FALSE.
File read limit (per scan)	The number of files read per sweep of the file directory
ackagent	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	Protocol on which to make acknowledgement copies (Currently only FILE is supported.)
to	Location for the acknowledgement; for example, `f:\fileout\ack.xml`.
Transform Type	Select the pre-built transform template from the drop-down box. All template names are prefixed with XML_to *documentType*.
Character Set Encoding	The character set encoding for inbound documents. For example, UTF-8. The default value is ISO-8859-1 US and Western Europe.

4. See "Common Service and Event Settings" on page 3-14 for information about selecting a schema and configuring tracing.

Using Batch Processing

To enable batch processing, select *BatchSplitter* from the Transform Type drop-down list.

The BatchSplitter preparses an entire EDI document and splits the document into individual transactions. Each transaction retains its Interchange Header/Trailer and Function Group Header/Trailer information. Once the BatchSplitter is finished splitting the EDI document, the transactions are ready to be transformed into XML. The following diagram illustrates how the BatchSplitter processes an EDI document:


File System Event
BatchSplitter
Interchange Header (ISA)
Functional Group Header (GS)
Transactional Set Header (ST)
Detail Segments
Transactional Set Trailer (SE)
Transactional Set Header (ST)
Detail Segments
Transactional Set Trailer (SE)
Functional Group Trailer (GE)
Interchange Trailer (IEA)
Interchange Header (ISA)
Functional Group Header (GS)
Transactional Set Header (ST)
Detail Segments
Transactional Set Trailer (SE)
Functional Group Trailer (GE)
Interchange Trailer (IEA)
Interchange Header (ISA)
Functional Group Header (GS)
Transactional Set Header (ST)
Detail Segments
Transactional Set Trailer (SE)
Functional Group Trailer (GE)
Interchange Trailer (IEA)

Before you create an event for your HIPAA application view and apply the BatchSplitter, you must know which types of HIPAA transactions (for example, 270, 271, and so on) are included in the EDI document before processing begins. If the EDI document contains various types of HIPAA transactions, you must create a unique event for each transaction and select the type of transaction from the Schema drop-down list.

FTP Event

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

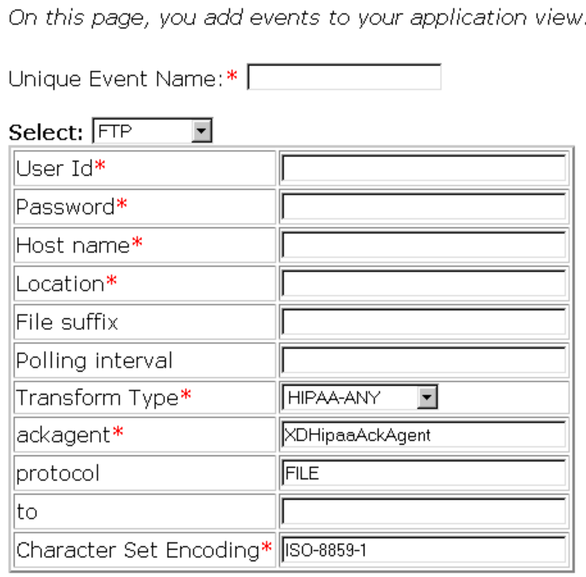
http://edocs.bea.com/wli/docs81/aiuser/index.html

In an FTP Event, the adapter picks up a file via FTP and passes it to your business process.

To configure an FTP Event:

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

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



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

Unique Event Name:*

Select: FTP

User Id*	
Password*	
Host name*	
Location*	
File suffix	
Polling interval	
Transform Type*	HIPAA-ANY
ackagent*	XDHipaaAckAgent
protocol	FILE
to	
Character Set Encoding*	ISO-8859-1

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

3. Enter the following information:

Table 3-7 FTP Event Parameters

Parameter	Description
User Id	The user name to log in to the FTP server
Password	The password for the FTP user
Host name	The name of the FTP host
Location	The directory where messages are received. DOS-style file patterns are valid for this parameter. You can specify a file pattern as well as a directory. For example, `c:\xyz\ab*cd` (without a file suffix) takes the file suffix from that parameter. If you use a pattern, files are selected based on the suffix and then the pattern. AB?CD selects ABxCD. AB*CD selects ABxxxCD.
File Suffix	File extension for the file event. This limits input files to those with the specified extensions. The "." is not required. The minus sign ("-") indicates that there is no extension. If the file extension is `zip`, the unzipped files must conform to the event schema, or they will fail. This function also works with transform configured.
Polling Interval	This is a time, expressed as xxH:xxM:xxS For example 1 hour, 2 minutes, and 3 seconds is: 1H:2M:3S The maximum 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 a worker thread cannot respond to a stop command. If this value is set to 0, the listener runs once and terminates. The default value is 2 seconds.
Transform Type	Select the pre-built transform template from the drop-down box. All template names are prefixed with XML_to *documentType*.
ackagent	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.

Table 3-7 FTP Event Parameters (Continued)

Parameter	Description
protocol	Protocol on which to make acknowledgement copies (Currently only FILE is supported.)
to	Location for the acknowledgement; for example, `f:\fileout\ack.xml`.
Character Set Encoding	The character set encoding for inbound documents. For example, UTF-8. The default value is ISO-8859-1 US and Western Europe.

4. See "Common Service and Event Settings" on page 3-14 for information about selecting a schema and configuring tracing.

HTTP Event

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

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

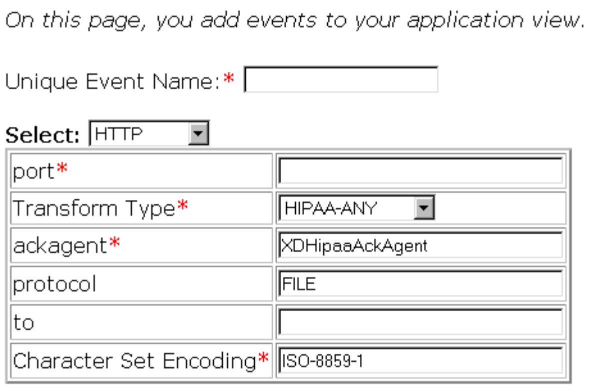
`http://edocs.bea.com/wli/docs81/aiuser/index.html`

In an HTTP Event, the adapter picks up a file via HTTP and passes it to a business process.

To configure an HTTP Event:

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

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


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

Unique Event Name:*

Select: HTTP

port*	
Transform Type*	HIPAA-ANY
ackagent*	XDHipaaAckAgent
protocol	FILE
to	
Character Set Encoding*	ISO-8859-1

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

3. Enter the following information:

Table 3-8 HTTP Event Parameters

Parameter	Description
port	The port where the adapter listens for the HTTP transfer
Transform Type	Select the pre-built transform template from the drop-down box. All template names are prefixed with XML_to *documentType*.
ackagent	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	Protocol on which to make acknowledgement copies (Currently only FILE is supported.)
to	Location for the acknowledgement; for example, `f:\fileout\ack.xml`.
Character Set Encoding	The character set encoding for inbound documents. For example, UTF-8. The default value is ISO-8859-1 US and Western Europe.

4. See "Common Service and Event Settings" on page 3-14 for information about selecting a schema and configuring tracing.

MQ Event

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

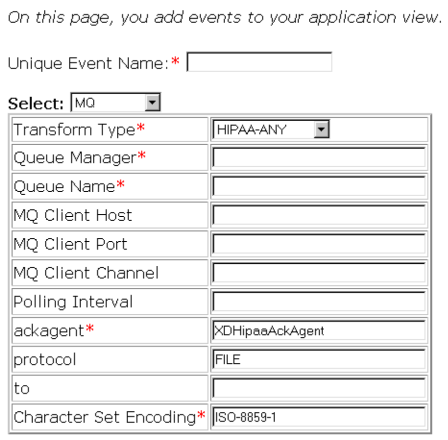
`http://edocs.bea.com/wli/docs81/aiuser/index.html`

In an MQ Event, the adapter picks up a file from a specific IBM MQSeries or WebSphere MQ queue and passes it to a business process.

To configure an MQ Event:

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

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



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

Unique Event Name:*

Select: MQ

Transform Type*	HIPAA-ANY
Queue Manager*	
Queue Name*	
MQ Client Host	
MQ Client Port	
MQ Client Channel	
Polling Interval	
ackagent*	XDHipaaAckAgent
protocol	FILE
to	
Character Set Encoding*	ISO-8859-1

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

3. Enter the following information:

Table 3-9 MQEvent Parameters

Parameter	Description
Transform Type	Select the pre-built transform template from the drop-down box. All template names are prefixed with XML_to *documentType*.

Table 3-9 MQEvent Parameters (Continued)

Parameter	Description
Queue Manager	The name of the MQ Queue Manager to be used
Queue Name	The name of the MQSeries or WebSphere MQ queue that the HIPAA system polls
MQ Client Host	For MQ Client only. The host where the MQ Server is located
MQ Client Port	For MQ Client only. The port number used to connect to an MQ Server
MQ Client Channel	For MQ Client only. The channel between an MQ Client and MQ Server
Polling Interval	The maximum wait interval (in the format `nnH:nnM:nnS`) between checks for new documents. The higher this value, the longer the interval, and the fewer system resources that are used. However, with a high value, 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	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	Protocol on which to make acknowledgement copies (Currently only FILE is supported.)
to	Location for the acknowledgement; for example, `f:\fileout\ack.xml`.
Character Set Encoding	The character set encoding for inbound documents. For example, UTF-8. The default value is ISO-8859-1 US and Western Europe.

4. See "Common Service and Event Settings" on page 3-14 for information about selecting a schema and configuring tracing.

TCP Event

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

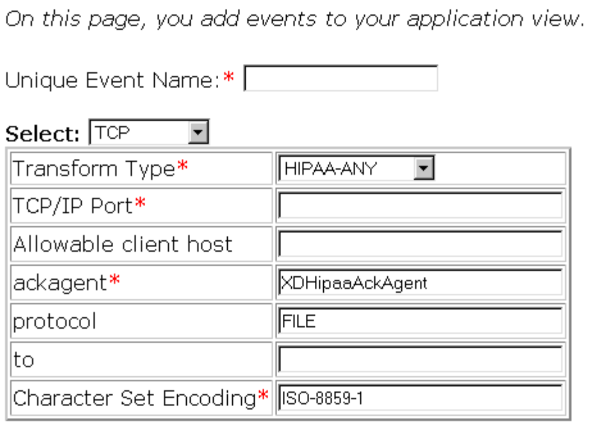
http://edocs.bea.com/wli/docs81/aiuser/index.html

In a TCP Event, the adapter picks up a file via TCP and passes it to a business process.

To configure a TCP Event:

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

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



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

Unique Event Name:*

Select: TCP

Transform Type*	HIPAA-ANY
TCP/IP Port*	
Allowable client host	
ackagent*	XDHipaaAckAgent
protocol	FILE
to	
Character Set Encoding*	ISO-8859-1

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

3. Enter the following information:

Table 3-10 TCP Event Parameters

Parameter	Description
Transform Type	Select the pre-built transform template from the drop-down box. All template names are prefixed with XML_to *documentType*.
TCP/IP Port	The port where the adapter listens for the TCP transfer
Allowable client host	The name or address of the client restricted to accessing this adapter
ackagent	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.

Table 3-10 TCP Event Parameters (Continued)

Parameter	Description
protocol	Protocol on which to make acknowledgement copies (Currently only FILE is supported.)
to	Location for the acknowledgement; for example, `f:\fileout\ack.xml`.
Character Set Encoding	The character set encoding for inbound documents. For example, UTF-8. The default value is ISO-8859-1 US and Western Europe.

4. See "Common Service and Event Settings" on page 3-14 for information about selecting a schema and configuring tracing.

Defining Event Connection Parameters

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

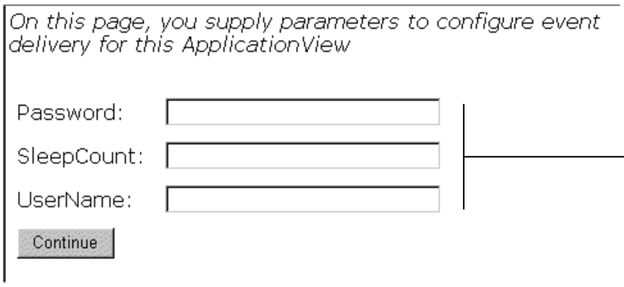
`http://edocs.bea.com/wli/docs81/aiuser/index.html`

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

To define event connection parameters:

1. In Connections area on the Application View Administration page, click Select/Edit.
2. In the Event Connection area, click Event to edit the default event connection.

   The Configure Event Delivery Parameters page appears.


On this page, you supply parameters to configure event delivery for this ApplicationView

Password:	
SleepCount:	
UserName:	

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

3. Enter the following information:

Table 3-11 Event Connection Parameters

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

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

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

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

   – Log errors and audit messages

   – Log warnings, errors, and audit messages

   – Log informational, warning, error, and audit messages

   – Log all messages

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

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

Table 3-12 Logging message categories

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

Testing Services

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

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

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

Note: You can test an application view only if it is deployed and only if it contains at least one event or service.

To test a service:

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

   The Test Services page appears.

2. In the Test Service window, copy the appropriate XML strings.

3. Click Test.

   The results appear in the Test Results window.

   If you are testing an MQ service, you can also check your MQSeries Explorer for results (select Queue Managers>*MyQueueManager*>Queues). If the service ran correctly, the queue now contains the message.

Testing Events Using a Service

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

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

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

Note: You can test an application view only if it is deployed and only if it contains at least one event or service.

To test an event:

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

   The Test Events page appears.

2. Click Service and select a service that triggers the event you are testing.

3. In the Time field, enter a reasonable period of time to wait, specified in milliseconds, before the test times out (One second = 1000 milliseconds. One minute = 60,000 milliseconds.).

4. Click Test and enter the XML string needed to trigger the service.

   The service is executed.

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

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

Testing Events Manually

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

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

To test an event manually:

1. In the Time field, enter a reasonable period of time to wait, specified in milliseconds, before the test times out (One second = 1000 milliseconds. One minute = 60,000 milliseconds.).
2. Click Test. The test waits for an event to trigger it.
3. Using the triggering HIPAA application, perform an action that executes the service that, in turn, tests the application view event.
   - If the test succeeds, the Test Result page appears. This page displays the event document from the application, the service input document, and the service output document.
   - If the test fails or takes too long, the Test Result page appears, showing a Timed Out message.

CHAPTER 4

Writing and Editing Rule Specification Files

A complete set of validation rules is supplied with 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:

About Rule Specification Files

The Rule Specification file is an XML document. You must have one file 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).

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. To learn more about writing your own rules in Java, see "About Rules In Java" on page 4-17.
- `<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 About the 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 About the 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.

About the Syntax for Writing Rules

The following table lists the general syntax for writing rules.

Table 4-1 Rules Writing Syntax

Rule Element	Attribute	Description
<using>	class=	The Java program class containing all <rule>s within the section, unless overridden by a class= attribute in the <rule> entry itself.
<rule>	tag=	Names the right-most parts of the tag to which this rule applies. The rule applies to any node of the document that meets the tag criteria. For example, 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.
<rule>	stag=	For HIPAA documents, this is a specification subsection tag. This tag is explained in "About Built-in HIPAA Rules" on page 4-4.

Table 4-1 Rules Writing Syntax (Continued)

Rule Element	Attribute	Description
<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.

About 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.

Using the 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.

About the Supported Designators

Table 4-2 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.	

Table 4-2 Supported Designators (Continued)

Condition Code		Meaning	Definition
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.

Using 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.

About the 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 4-3 checkDTM

Attribute	Meaning
cd	See "Using the Condition Designator (cd=)" on page 4-5.
code	The value of the DTM01 element within the DTM must be one of the codes in the list. For example: <rule … code="405, 412" …> The addition of a /N after a code means that the date field must be null. Otherwise the date must not be null.

checkDTP

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

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

Table 4-4 checkDTP

Attribute	Meaning
cd	See "Using the Condition Designator (cd=)" on page 4-5.
code	An if/then list for the 01/02/03 type/format/date-time subelements. See "Using If/Then Date Format Rules" on page 4-6 for format of an if/then list.

checkDMG

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

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

Table 4-5 checkDMG

Attribute	Meaning
cd	See "Using the Condition Designator (cd=)" on page 4-5.

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 4-6 checkQTY

Attribute	Meaning
cd	See "Using the Condition Designator (cd=)" on page 4-5.
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 4-7 checkCD

Attribute	Meaning
cd	See "Using the Condition Designator (cd=)" on page 4-5.

isN

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

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

Table 4-8 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 4-9 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 4-10 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 4-11 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 4-12 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 4-13 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 *stag* 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 4-14 checkUsage

Attribute	Meaning

Code	`The if/then validation criteria in the form`
	`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 := ! | =`

<action> codes	
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.

An if clause’s <value> of ? means that the then side applies regardless of the code value.

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 4-15 isCDate

Attribute	Meaning
`code`	List of valid qualifiers or if/then list if *type*= 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 *type*= 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 4-16 checkList

Attribute	Meaning
`code`	One of the following: 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().

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 4-19.

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 4-17 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 4-18 segXO

Attribute	Meaning
a	First value.

Table 4-18 segXO

Attribute	Meaning
`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 4-19 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 4-20 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 4-21 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 4-22 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 4-23 tranBal

Attribute	Meaning
No attributes	No parameters

About 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 4-24 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 node.getValue() return if the tag contained a subfield address (for example, tag=x:2).
`Attributes`	A HashMap of rule attributes. The rule method can check for any attributes that it desires. A HashMap is a fast implementation of a Hashtable that does not serialize.

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 XDException
    {
        trace(XD.TRACE_DEBUG, "specialRule called with parms: " +
              node.getFullName() + ", " + attributes.toString());
        String testValue = (String)attributes.get(“value”);
        if (value.equals(testValue) )
        {
 node.setAssociatedVector(new XDEDIError(4, 0, error,”explanation”));
            throw new XDException(XD.RULE, XD.RULE_VIOLATION,"node value
"+value+” is not 'Value'");
        }
    }
}
```

Rule violations should throw an XDException describing the violation.

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

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.

Loading a Java File

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

```
import com.ibi.edaqm.*;
import java.util.*;
import java.io.*;
/**
 * A rule list handler is a routine called to enable users search lists during
execution
 * or the checkList rule. checkList() is a generally available rule to test
whether the
 * contents of a document field are valid.  The rule list handler is invoked when
 * the code= attribute indicates the name of a coder routine rather than a simple
list.<P>
 * For example, <I>code="@list1"</I> will cause the search routine of the list1
class to
 * be invoked.<P>
 * The file read by this procedure consists of tokens separated by new line,
white space or commas.
 */
public class XDRuleListFile extends XDRuleList
{
        String[] list;
        ArrayList al = new ArrayList(127);
        public XDRuleListFile()
        {
        }
        /**
         * The init method is called when a rule is loaded. It can perform any
necessary
        * initialization, and can store any persistent information in the object
store.
         *
        * @param parms  Array of parameter string passed within the start command
init-name(parms).
         */
        public void init(String[] parms) throws XDException
        {
            if (parms == null)
            {
                    throw new XDException(XD.RULE, XD.RULE_SYNTAX, "no parms
    sent to " + name);
                }
                try
                {
                    File  f = new File(parms[0]);
                    FileInputStream fs = new FileInputStream(f);
                    long len = f.length();
                    byte[] b = new byte[(int)len];
                    fs.read(b);
                    fs.close();
                    String data = new String(b);
                    StringTokenizer st = new StringTokenizer(data, ", " +
    XD.NEWLINE);
                    while (st.hasMoreTokens())
                    {
                        String part = st.nextToken();
                        al.add(part);
                    }
                }
                catch (FileNotFoundException e)
                {
                    throw new XDException(XD.RULE, XD.RULE_SYNTAX, "list file
"+parms[0] + " not found");
            }
            catch (IOException eio)
            {
            throw new XDException(XD.RULE, XD.RULE_SYNTAX, eio.toString());
            }
        }
        /**
        * The term() method is called when the worker is terminated. It is
NOT guaranteed
        * to be call, and applications should not rely upon this method to
update data bases or
         * perform other critical operations.
         */
        public void term()
        {
        }
        /**
         * Search the given value to determine whether it is in the list.
         *
         * @param value  String to test against the list
         * @return true if found, false otherwise
         */
        public boolean search(String value)
        {
            return al.contains(value);
        }
}
```

CHAPTER 5

Functional Acknowledgement Handling

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

- About 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 4, “Writing and Editing Rule Specification Files.”

About 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 4, "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.

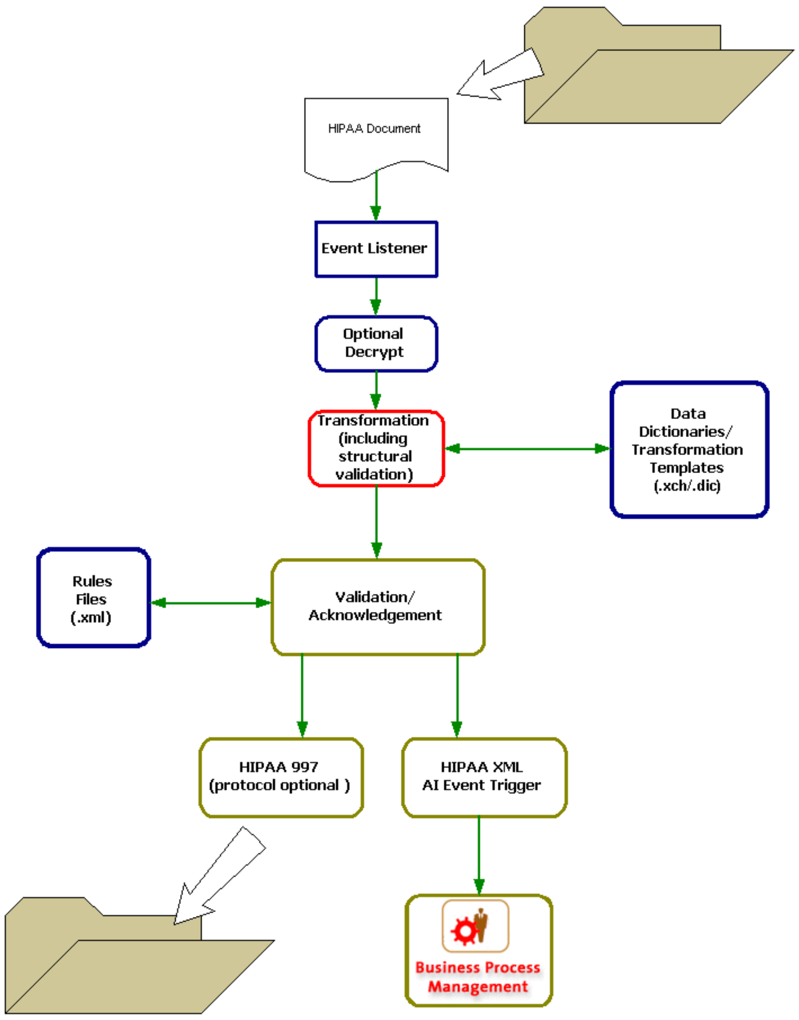
- Acknowledgement Agent

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.

About 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 5-1 Document Lifecycle

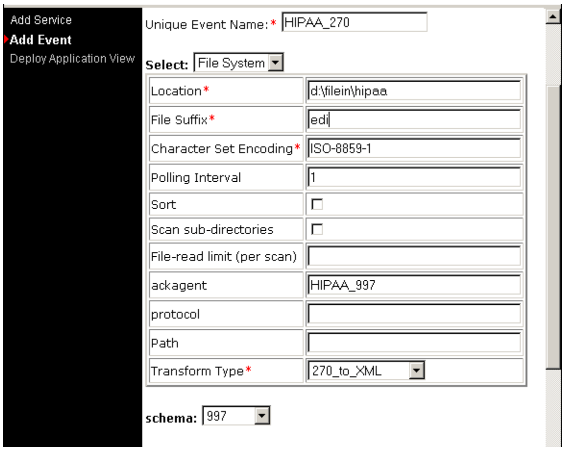

HIPAA Document
Event Listener
Optional Decrypt
Transformation (including structural validation)
Data Dictionaries/ Transformation Templates (.xch/.dic)
Rules Files (.xml)
Validation/ Acknowledgement
HIPAA 997 (protocol optional)
HIPAA XML AI Event Trigger
Business Process Management

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 4, "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.

Using the 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`).



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	☐
Scan sub-directories	☐
File-read limit (per scan)	
ackagent	HIPAA_997
protocol	
Path	
Transform Type*	270_to_XML

schema: 997

About 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 5-1 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 5-2 Acknowledgement Result Schema

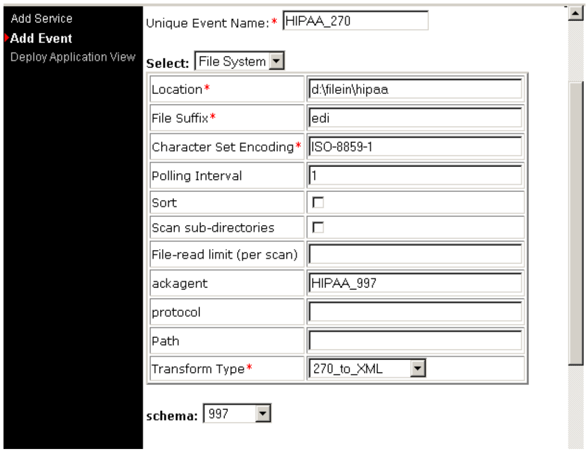
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
       <xs:element name="Error" type="xs:string"/>
       <xs:element name="HIPAAack">
              <xs:complexType>
                     <xs:sequence>
                            <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.



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	☐
Scan sub-directories	☐
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. Publish the application view and create a workflow in WebLogic Workshop.

Index