*SeeBeyond ICAN Suite*

# HIPAA OTD Library User's Guide

*Release 5.0*

Version 20040604052556.

# Contents

# List of Tables

# List of Figures

Chapter 1

# Introduction

This chapter provides an introduction to the HIPAA OTD Library User's Guide.

The Health Insurance Portability and Accountability Act of 1996 (HIPAA) is a law that, among other things, mandates certain standards specifically for the health care industry. For transactions related to health care, HIPAA uses a customization of X12. For pharmaceutical transactions, the HIPAA standard uses NCPDP (National Council for Prescription Drug Programs) transactions.

This book includes an overview of HIPAA, and then specific information relating to the installation and contents of SeeBeyond's HIPAA OTD Library.

A general overview of X12 is also included in **Appendix A**.

## 1.1 Overview

Each of the eGate™ Object Type Definitions (OTD) libraries contains sets of pre-built structures for industry-standard formats.

The HIPAA OTD library is a feature of the SeeBeyond™ eBusiness Integration Suite, and contains message definitions for HIPAA messages. This document gives a brief overview of HIPAA and the HIPAA message structures provided with the eGate Integrator, and provides information on installing and using the HIPAA OTD libraries.

## 1.2 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the SeeBeyond ICAN Suite (such as eGate Integrator and eXchange Integrator), to have familiarity with Windows operations and administration, and to be thoroughly familiar with Microsoft Windows graphical user interfaces.

# 1.3 Supported Operating Systems

The eGate HIPAA OTD Library is available on the following platforms:

- Microsoft Windows 2000, Windows XP, and Windows 2003
- Sun Solaris 8 and Solaris 9
- IBM AIX 5L Version 5.1 and AIX 5L Version 5.2
- HP-UX 11.0 and HP-UX 11i (PA-RISC)
- HP Tru64 UNIX Version 5.1A
- Red Hat Linux 8 (Intel Version) and Linux Advanced Server 2.1 (Intel version)

***Note:*** ***UNIX Systems****—This guide uses the backslash ("\") as the separator within path names. If you are working on a UNIX system, make the appropriate substitutions.*

# 1.4 Writing Conventions

The following writing conventions are observed throughout this document.

**Table 1** Writing Conventions

| Text | Convention | Example |
|---|---|---|
| Names of buttons, files, menus and menu items, icons, parameters, variables, methods, and objects | **Bold** text | ▪ Click **OK** to save and close.<br>▪ Select the **logicalhost.exe** file.<br>▪ On the **File** menu, click **Exit**.<br>▪ Enter the **timeout** value.<br>▪ Use the **getClassName()** method. |
| Command-line arguments, code samples | `Fixed` font. Variables are shown in ***`bold italic`***. | `bootstrap -f -p` ***`password`*** |
| Hypertext links | **Blue** text | ▪ For more information, see **"Writing Conventions" on page 10**.<br>▪ **http://www.seebeyond.com** |

## Additional Conventions

### Windows Systems

For the purposes of this guide, references to "Windows" will apply to Microsoft Windows Server 2003, Windows XP, and Windows 2000.

### Path Name Separator

This guide uses the backslash ("\") as the separator within path names. If you are working on a UNIX or HP NonStop system, please make the appropriate substitutions.

# 1.5 Supporting Documents

The following SeeBeyond documents provide additional information that might prove useful to you.

- *X12 OTD Library User's Guide*
- *eGate Integrator Installation Guide*
- *UN/EDIFACT OTD Library User's Guide*

# 1.6 SeeBeyond Web Site

The SeeBeyond Web site is your best source for up-to-the-minute product news and technical support information. The site's URL is:

http://www.seebeyond.com

Chapter 2

# HIPAA Overview

This chapter provides the following information:

- An overview of HIPAA, including general information
- A list of the specific transactions that comprise the HIPAA standard
- The structure of HIPAA envelopes, data elements, and syntax.

## 2.1 Introduction to HIPAA

The following sections provide an introduction to HIPAA and the message structures that are included in the HIPAA OTD Library.

### 2.1.1. What Is HIPAA?

HIPAA is an acronym for the Health Insurance Portability and Accountability Act of 1996. This Act is designed to protect patients. Among other things, it makes specifications affecting standards of treatment and privacy rights. It provides a number of standardized transactions that can be used for such things as a healthcare eligibility inquiry or a healthcare claim. HIPAA legislates that all of the healthcare industry will be on the same implementation timetable. All institutions performing electronic healthcare insurance transactions must implement these standardized transactions by October 2002 unless they have obtained an extension to October 2003.

More transactions will be added for later implementation.

HIPAA legislation mandates, among other items:

- Standards for maintaining patient confidentiality and helping to ensure privacy by mandating security options.
- Use of a national payer ID.
- Use of a national provider ID.

HIPAA regulations affect many organizations dealing with the medical industry, such as:

- Automated clearing houses (ACHs)
- Transaction processors
- Value-added networks (VANs)

- Payers
- Health insurance providers
- Provider management organizations

For transactions relating to such things as health care claims, the HIPAA standard uses a range of customized X12 transactions.

For transactions relating to prescriptions, HIPAA uses NCPDP (National Council for Prescription Drug Programs) transactions.

The HIPAA X12 standard, being based on X12, includes loops, segments, and data elements. In addition, it mandates consistent use of these elements across all HIPAA implementation guides.

The X12 portion of the HIPAA OTD Library provides Object Type Definitions for all nine standard X12 transactions that have been adopted by HIPAA, as listed in Table 2.

These transactions are based on the October 1997 X12 standard; that is, Version 4, Release 1, Sub-release 0 (004010).

**Table 2** HIPAA X12 Transactions

| Number | Name |
|---|---|
| 270 | Eligibility Coverage or Benefit Inquiry |
| 271 | Eligibility Coverage or Benefit Information |
| 276 | Health Care Claim Status Request |
| 277 | Health Care Claim Status Notification |
| 278 | Two versions: Health Care Services Review Information and Request for Review/Response to Request |
| 820 | Payment Order Remittance Advice |
| 834 | Benefit Enrollment and Maintenance |
| 835 | Health Care Claim Payment Advice |
| 837 | Health Care Claim (three versions: Professional, Dental, and Institutional) |

The NCPDP portion of the HIPAA OTD Library provides request and response transactions for all the HIPAA-approved NCPDP transaction codes, as listed in Table 3.

**Table 3** NCPDP-HIPAA Transaction Codes

| Code | Transaction Name |
|---|---|
| E1 | Eligibility Verification |
| B1 | Billing |
| B2 | Reversal |
| B3 | Rebill |
| P1 | Prior Authorization Request and Billing |
| P2 | Prior Authorization Reversal |
| P3 | Prior Authorization Inquiry |

**Table 3** NCPDP-HIPAA Transaction Codes (Continued)

| Code | Transaction Name |
|---|---|
| P4 | Prior Authorization Request Only |
| N1 | Information Reporting |
| N2 | Information Reporting Reversal |
| N3 | Information Reporting Rebill |
| C1 | Controlled Substance Reporting |
| C2 | Controlled Substance Reporting Reversal |
| C3 | Controlled Substance Reporting Rebill |

***Note:*** *While the HIPAA OTD Library includes both X12 and NCPDP OTDs, this document primarily discusses the HIPAA X12 OTDs. For more information about the NCPDP-HIPAA OTD Library, see the NCPDP OTD Library User's Guide.*

## 2.1.2. Trading Partner Agreements

Although the regulations mandated by HIPAA are very strict and specific, it is still important to have trading partner agreements for individual trading relationships.

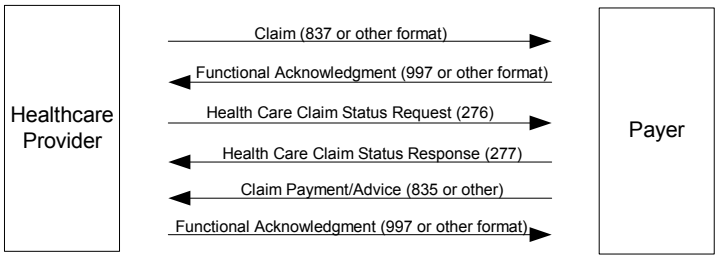
Following the HIPAA standard ensures that transactions comply with the regulations mandated by the government. HIPAA requirements are completely described in the HIPAA implementation guide for each transaction, and must not be modified by a trading partner.

However, there is room for negotiation in terms of the specific processing of the transactions in each trading partner's individual system. It is normal for trading partners to have individual agreements that supplement the standard guides. The specific processing of the transactions in each trading partner's individual system might vary between sites. Because of this, additional documentation that provides information about the differences is helpful to the site's trading partners and simplifies implementation. For example, while a certain code might be valid in an implementation guide, a specific trading partner might not use that code in transactions. It would be important to include that information in a trading partner agreement.

## 2.1.3. Sample Scenario

An example of a HIPAA X12 transaction exchange between a health care provider and a payer is shown in Figure 1.

**Figure 1** Sample HIPAA Transaction Exchange




## 2.1.4. Batch and Real-Time Transactions

The HIPAA standard supports the sending and receiving of messages in both batch and real-time (interactive) modes.

### Batch

In batch mode, transactions are grouped together and multiple transactions are sent in a single message. The batch can either go directly to the receiver or via a clearing house. The connection does not remain open while the receiver processes the messages. If there is an expected response transaction (for example, a 271 in response to a 270) the receiver creates the response transaction offline and then sends it.

### Real Time

If a transaction is processed in real time, it is sent individually. Transactions that require an immediate response are normally sent in real time. In real-time mode, the sender sends the request transaction, either directly or through a clearing house, and the connection is kept open while the receiver processes the transaction and returns a response transaction. Response times are typically no more than one minute, and often less.

In real-time mode, the receiver must send a response; either the expected response transaction, such as a 271 in response to a 270, or a standard acknowledgment such as the 997.

## 2.1.5. Data Overview

HIPAA X12 transactions all use the standard components of the X12 standard, covered in **Appendix A, "ASC X12 Overview" on page 51**.

Specifically, the transactions use the following elements:

- Segments
- Data elements
- Looping structures

### 2.1.6. Acknowledgment

The HIPAA X12 transactions either have specific response transactions or use the standard 997 Functional Acknowledgment.

The 997 Functional Acknowledgment is used by the following transactions:

- 837 (sent by the payer to acknowledge claim receipt)
- 277 (sent by the provider to acknowledge receipt of a Health Care Payer Unsolicited Claim Status request)
- 277 (sent by the provider to acknowledge receipt of a Health Care Claim Request for Additional Information)
- 835 (sent by the provider to acknowledge receipt of a Health Care Claim Payment/ Advice notification)

## 2.2 Additional Information

For more information on HIPAA, visit the following Web sites:

- **http://www.cms.hhs.gov**
- **http://www.hipaa-dsmo.org**
- **http://www.wedi.org/**
- **http://www.claredi.com/**
- **http://aspe.os.dhhs.gov/admnsimp/**

For more information on NCPDP, visit the official NCPDP Web site at this address:

- **http://www.ncpdp.org/**

***Note:*** *This information is correct at the time of going to press; however, SeeBeyond has no control over these sites. If you find the links are no longer correct, use a search engine to search for* ***HIPAA****.*

Chapter 3

# HIPAA OTD Library Installation

This chapter provides information on installing the HIPAA OTD library, and shows the resulting Project Explorer Tree for the OTDs. It includes general installation information and step-by-step installation instructions.

Use of the HIPAA OTD Library requires installation of at least one of the available .sar files found on the installation CD ROM.

Licensed files that are available for installation include:

- HIPAA_2000_Standard_OTD_Lib.sar
- HIPAA_2000_Addenda_OTD_Lib.sar

## 3.1 HIPAA OTD Libaries

Installation of the complete HIPAA OTD Library creates two subdirectories under the **SeeBeyond > OTD Library > HIPAA** hierarchy. Each subdirectory contains twenty four OTDs and include the following X12 transactions.

- **2000_Standard** – includes HIPAA X12 transactions for the May 1999 and May 2000 standard, and NCPDP transactions.
- **2000_Addenda** – includes HIPAA X12 transactions that are February 2003 amendments to the May 2003 standard (the 00401010A1 Addenda)

Some additional points to note about the HIPAA transactions:

- The OTDs only accept messages with all the envelope segment information.
- Messages can be batched; however, all the messages in one functional group must be of the same message type.
- Apart from their use by eXchange, OTDs can also be used independently for eGate collaborations not associated with eXchange.

# 3.2 Installation Procedure

The steps for installing the HIPAA OTD Library are the same as for other products in the ICAN Suite. You can find general product installation instructions in the *ICAN Suite Installation Guide*, which is available on the product installation CD-ROM and can also be accessed via Enterprise Manager (Documentation tab).

## 3.2.1. Uploading files to the Repository

### Before you begin

- A Repository server must be running on the machine where you will be uploading the product files.
- You must have already uploaded **eGate.sar** (for either eGate 5.0.4 or eGate 5.0.3 with appropriate ESRs), and you must have already uploaded a **license.sar** file that includes a license for the X12 OTD library product.

### To upload product files to the Repository

1. On a Windows machine, start a Web browser and point it at the machine and port where the Repository server is running:

   `http://<hostname>:<port>`

   where
   - *<hostname>* is the name of the machine running the Repository server.
   - *<port>* is the starting port number assigned when the Repository was installed.

   For example, the URL you enter might look like either of the following:

   ```
   http://localhost:12001
   http://serv1234.company.com:19876
   ```

2. In the Enterprise Manager **SeeBeyond Customer Login** page, enter your username and password.
3. When Enterprise Manager responds, click the **ADMIN** tab.
4. In the ADMIN page, click **Browse**.
5. In the **Choose file** dialog, click **ProductsManifest.xml**, and then click **Open**.
6. In the ADMIN page, click **Submit**.

   The lower half of the ADMIN page lists the product files you are licensed to upload.
7. In the Products column, find the **HIPAA_2000_Stardard_OTD_Lib** product, and then click the **Browse** button for it.
8. In the **Choose file** dialog, click the corresponding **HIPAA_2000_Standard_OTD_Lib.sar** file, and then click **Open**.
9. Repeat the previous two steps for the **HIPAA_2000_Addenda_OTD_Lib** product.

***Note:*** *SMEWebServices.sar is required for such features as encryption/decryption, signature verification, certificate authentication, and nonrepudiation.*

10. In the ADMIN page, click **Upload Now**.

## 3.2.2. Refreshing Enterprise Designer

### Before you begin

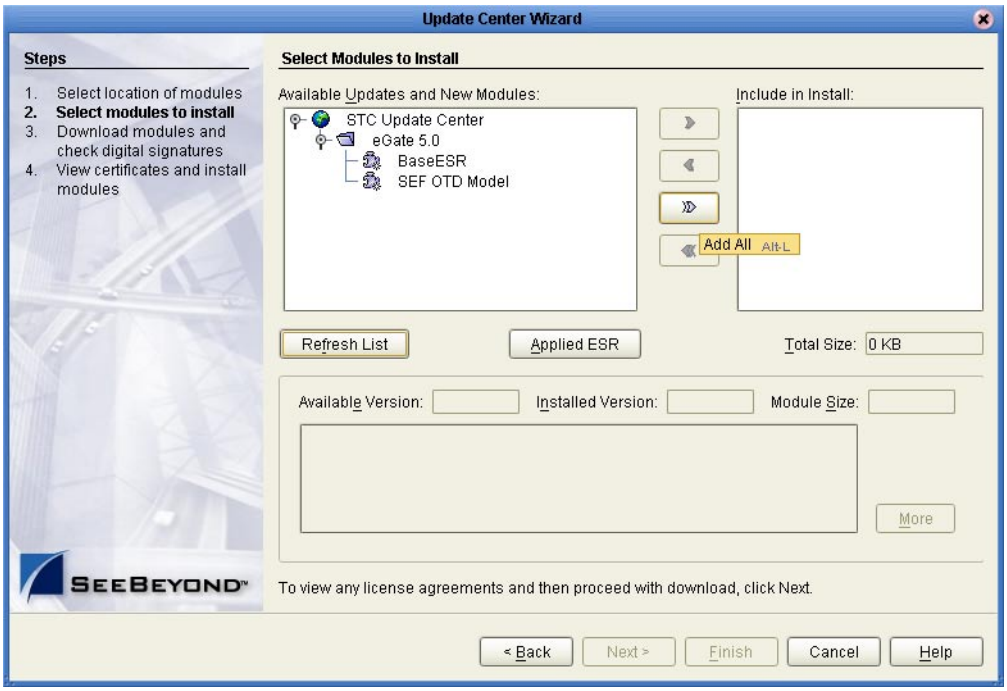
- You must have already downloaded and installed Enterprise Designer, and a Repository server must be running on the machine where you uploaded the HIPAA OTD Library product files.

### To refresh an existing installation of Enterprise Designer

1. Start Enterprise Designer.
2. On the **Tools** menu, click **Update Center**.

   The Update Center shows a list of components ready for updating. See Figure 2.

**Figure 2** Update Center Wizard: Select Modules to Install




3. Click **Add All** (the button with a doubled chevron pointing to the right).

   All modules move from the Available/New pane to the **Include in Install** pane.
4. Click **Next** and, in the next window, click **Accept** to accept the license agreement.
5. When the progress bars indicate the download has ended, click **Next**. Review the certificates and installed modules, and then click **Finish**. When prompted to restart Enterprise Designer, click **OK**.

   When Enterprise Designer restarts, the installation of the HIPAA OTD Library is complete.

If you need help on details of product installation, see the *SeeBeyond ICAN Suite Installation Guide*.

# 3.3 HIPAA Library OTDs

This section explains and provides a cross-reference for OTD files found in the HIPAA OTD Library.

The files include Object Type Definitions for the May 2000 and May 1999 HIPAA standards and for NCPDP Batch and Telecom transactions. May 1999 files are included for backwards compatibility only and do not include the more comprehensive validations of the May 2000 files.

Transactions modified according to rules published in the Federal Register on February 20, 2003 are installed to the same location as the May 2000 files.

## 3.3.1. Understanding HIPAA OTD Names

The names for the HIPAA OTDs are designed to assist you in quickly locating the file you want. The name for each transaction OTD is composed of the same set of elements in the same sequence.

Because Addenda have been created for each of the X12 Implementation Guides adopted for use under HIPAA and published in May, 2000, naming conventions differentiate between the original file set of May, 2000 and the Addenda published in February, 2003.

The federal Health and Human Services web site (www.cms.hhs.gov) describes the changes of 2003 as follows: "This final rule modifies a number or the electronic transactions and code sets adopted as national standards under HIPAA, and eliminates the NDC code set as the standard for all providers except retail pharmacies."

The names are constructed as follows:

- **x12_**

  Identifies the name of the standard used, followed by underscore

- **004010X092_**

  Identifies the HIPAA reference number for the transaction—which includes the X12 version—followed by an underscore in the original set or "A1" and then an underscore in the February 2003 Addenda set. The "92" represents a two-digit number unique to each transaction type. It can also be 91, 93, 94, 95, 96, 97, or 98).

- **00_**

  Identifies the year (00_ for 2000 files and Addenda files).

- **hipaa_xxx_**

  Identifies the HIPAA OTD, followed by the transaction ID, and then an underscore. For 278 and 837 transactions, the format is "hipaaA1_278," where "A1" represents a transaction sub-type and "278" or "837" is the transaction type).

  Transactions of type 837 (health care claims) are differentiated by Qn appended to the "hipaa" string, where n is a value of 1, 2, or 3, as follows:

  - Q1: type 837p (professional)

- Q2: type 837d (dental)
- Q3: type 837i (institutional)

For more information about ASC X12 and its subcommittees, see **www.x12.org**

***Note:*** *"A1" following the string "hipaa" has a different meaning than "A1" following the ten-digit HIPAA reference number. In the first case, A1 identifies a transaction sub-type. In the second case, "A1" identifies an Addendum file, one of several Addenda to the May 2000 standard.*

- **Abbreviation**

  Identifies the transaction name; for example, HealCareClaiPaym for Health Care Claim Payment

## 3.3.2. 2000_Addenda OTDs

The following OTDs are found within the 2000_Addenda subdirectory.

**Table 4** 2000_Addenda OTDs

| OTD Name |
|---|
| x12_004010X061A1_00_hipaa_820_PaymOrdeAdvi |
| x12_004010X061A1_00_hipaa_820_PaymOrdeAdvi_Full |
| x12_004010X091A1_00_hipaa_835_HealCareClaiPaym |
| x12_004010X091A1_00_hipaa_835_HealCareClaiPaym_Full |
| x12_004010X092A1_00_hipaa_270_EligCoveOrBeneInqu |
| x12_004010X092A1_00_hipaa_270_EligCoveOrBeneInqu_Full |
| x12_004010X092A1_00_hipaa_271_EligCoveOrBeneInfo |
| x12_004010X092A1_00_hipaa_271_EligCoveOrBeneInfo_Full |
| x12_004010X093A1_00_hipaa_276_HealCareClaiStatRequ |
| x12_004010X093A1_00_hipaa_276_HealCareClaiStatRequ_Full |
| x12_004010X093A1_00_hipaa_277_HealCareClaiStatNoti |
| x12_004010X093A1_00_hipaa_277_HealCareClaiStatNoti_Full |
| x12_004010X094A1_00_hipaaA1_278_HealCareServReviInfo |
| x12_004010X094A1_00_hipaaA1_278_HealCareServReviInfo_Full |
| x12_004010X094A1_00_hipaaA3_278_HealCareServReviInfo |
| x12_004010X094A1_00_hipaaA3_278_HealCareServReviInfo_Full |
| x12_004010X095A1_00_hipaa_834_BeneEnroAndMain |

**Table 4** 2000_Addenda OTDs

| OTD Name |
|---|
| x12_004010X095A1_00_hipaa_834_BeneEnroAndMain_Full |
| x12_004010X096A1_00_hipaa_q3_837_HealCareClai |
| x12_004010X096A1_00_hipaa_q3_837_HealCareClai_Full |
| x12_004010X097A1_00_hipaa_q2_837_HealCareClai |
| x12_004010X097A1_00_hipaa_q2_837_HealCareClai_Full |
| x12_004010X098A1_00_hipaa_q1_837_HealCareClai |
| x12_004010X098A1_00_hipaa_q1_837_HealCareClai_Full |

### 3.3.3. 2000_Standard OTDs

The following OTDs are found within the 2000_Standard subdirectory.

**Table 5** 2000_Standard OTDs

| OTD Name |
|---|
| x12_004010X061_00_hipaa_820_PaymOrdeAdvi |
| x12_004010X061_00_hipaa_820_PaymOrdeAdvi_Full |
| x12_004010X091_00_hipaa_835_HealCareClaiPaym |
| x12_004010X091_00_hipaa_835_HealCareClaiPaym_Full |
| x12_004010X092_00_hipaa_270_EligCoveOrBeneInqu |
| x12_004010X092_00_hipaa_270_EligCoveOrBeneInqu_Full |
| x12_004010X092_00_hipaa_271_EligCoveOrBeneInfo |
| x12_004010X092_00_hipaa_271_EligCoveOrBeneInfo_Full |
| x12_004010X093_00_hipaa_276_HealCareClaiStatRequ |
| x12_004010X093_00_hipaa_276_HealCareClaiStatRequ_Full |
| x12_004010X093_00_hipaa_277_HealCareClaiStatNoti |
| x12_004010X093_00_hipaa_277_HealCareClaiStatNoti_Full |
| x12_004010X094_00_hipaaA1_278_HealCareServReviInfo |
| x12_004010X094_00_hipaaA1_278_HealCareServReviInfo_Full |
| x12_004010X094_00_hipaaA3_278_HealCareServReviInfo |
| x12_004010X094_00_hipaaA3_278_HealCareServReviInfo_Full |
| x12_004010X095_00_hipaa_834_BeneEnroAndMain |

**Table 5** 2000_Standard OTDs

| OTD Name |
|---|
| x12_004010X095_00_hipaa_834_BeneEnroAndMain_Full |
| x12_004010X096_00_hipaa_q3_837_HealCareClai |
| x12_004010X096_00_hipaa_q3_837_HealCareClai_Full |
| x12_004010X097_00_hipaa_q2_837_HealCareClai |
| x12_004010X097_00_hipaa_q2_837_HealCareClai_Full |
| x12_004010X098_00_hipaa_q1_837_HealCareClai |
| x12_004010X098_00_hipaa_q1_837_HealCareClai_Full |

## 3.4 After Installation

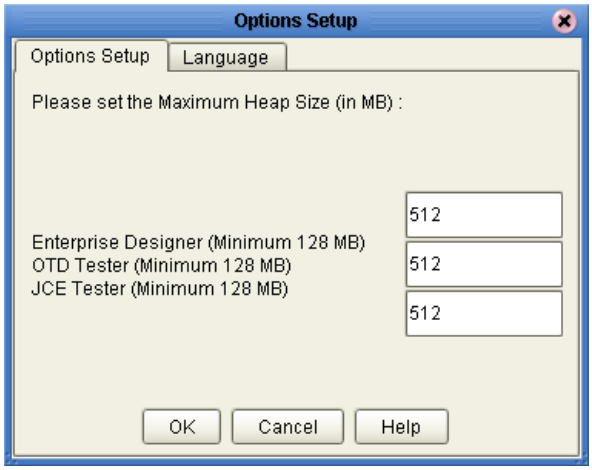
The HIPAA Library must be installed and incorporated into a Project before it can perform its intended functions. See the *eGate Integrator User's Guide* for more information on incorporating the OTDs into an eGate Project.

### 3.4.1. Increasing the ICAN Enterprise Designer Heap Size

Due to the size of the HIPAA OTD Libraries, the Enterprise Designer **Heap Size** may need to be increased prior to using eGate. A Heap Size that is not increased may result in an **OutOfMemoryError** message. To increase the heap size in Enterprise Designer, you must:

1. From the Enterprise Designer Menu bar, click **Tools** and select **Options**. The **Options Setup** dialog box appears.
2. Increase the configured heap size for the Enterprise Designer, OTDTester, and JCE Tester to 512 MB as displayed below.

**Figure 3** Options Setup window




3 Click **OK** to close the Options Setup window, then close and restart Enterprise Designer to allow your changes to take effect.

If an **OutOfMemoryError** message occurs while trying to open Enterprise Designer, the heap size settings may be changed prior to starting the program. In this case, change the settings in the **heapSize.bat** file, located in <*ICAN_Home*>\edesigner\bin (where <*ICAN_Home*> is the ICAN install directory). Open heapSize.bat with a text editor and change the heap size settings from **128** to **512**. Save the file and restart Enterprise Designer.

Chapter 4

# Working with HIPAA OTDs

This chapter provides information on additional features built into the HIPAA OTDs, and includes instructions on working with and testing the OTDs. This chapter also provides information on using the custom Java methods provided within the OTDs, and other general information about using the HIPAA OTD Library.

To test that your data is being mapped correctly by the OTD and that the data is valid based on definitions and business rules, you can run validation within the Java Collaboration Editor.

**Chapter Topics**

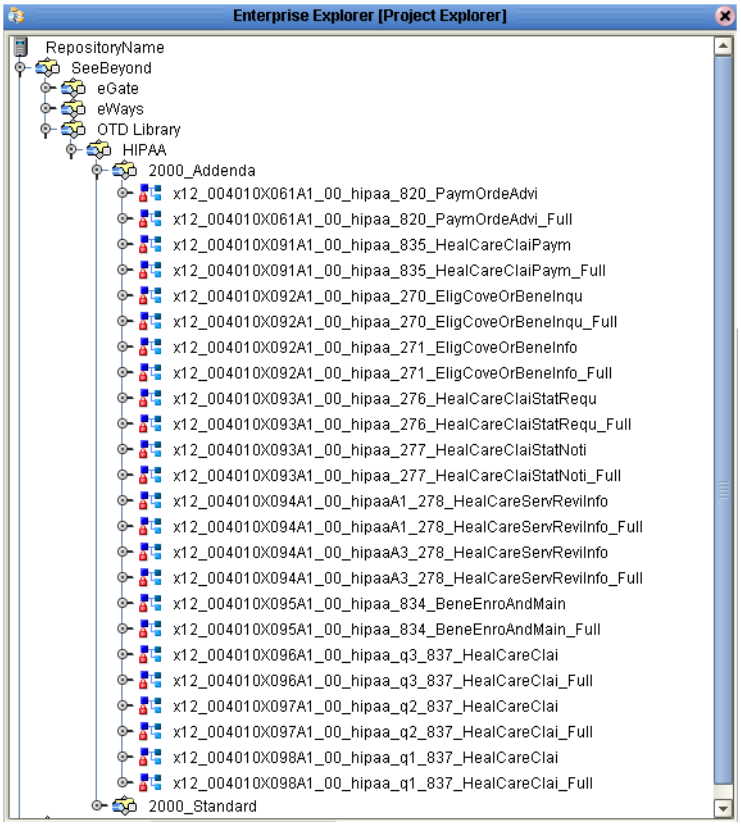


## 4.1 Viewing a HIPAA OTD with the OTD Editor

The installed HIPAA OTD Library is accessible from the SeeBeyond Enterprise Designer. To open a HIPAA OTD using the OTD Editor, you must:

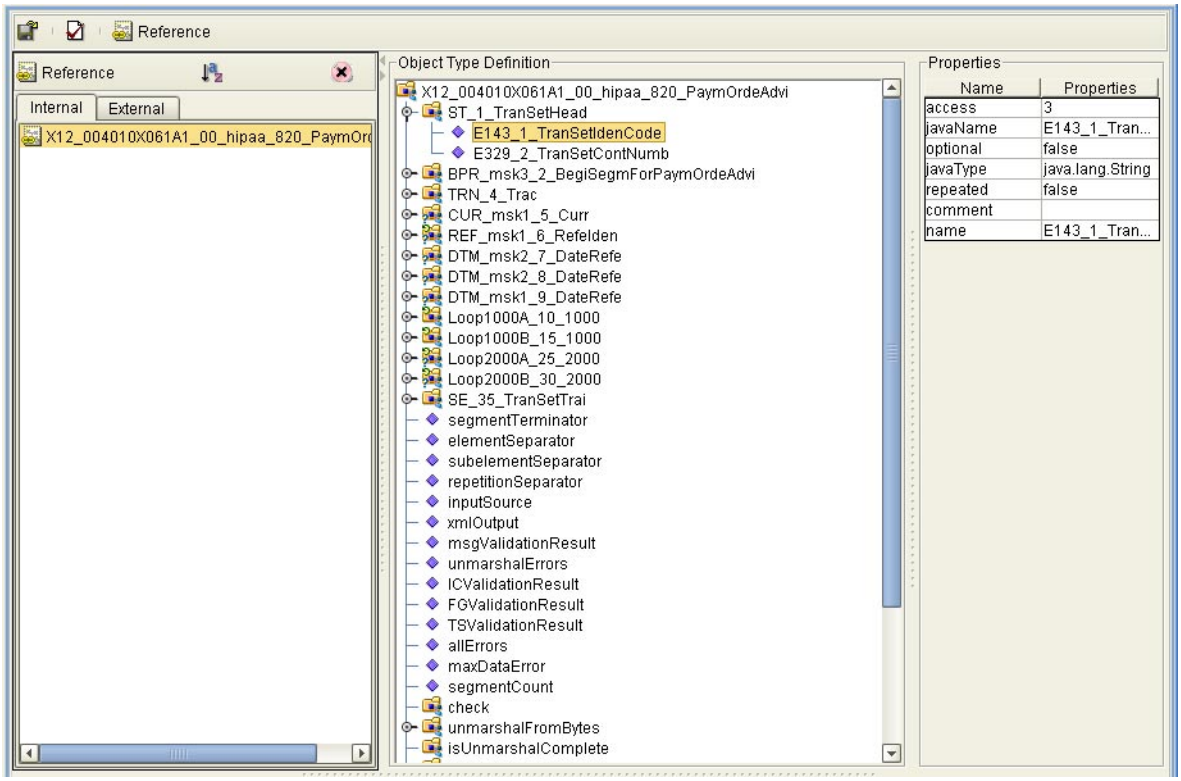
1. Expand the OTD Library node from Enterprise Explorer's Project Explorer tree, and select the subdirectory you want to view. For this example, the 2000_Addenda subdirectory is opened.

**Figure 4** 2000_Addenda OTDs




2 Double-click one of the listed OTDs. The OTD Editor appears displaying the selected OTD. You can expand or contract a node by double-clicking a node name or by single-clicking the icon handle next to the node name, as seen in Figure 5 below.

**Figure 5** The OTD Editor - Expanded Node




3 OTD properties are viewed and edited from the Properties pane in the upper right section of the OTD Editor. Properties for each node appears in this pane when selected.

## 4.2 Setting the Delimiters

HIPAA OTDs must include some way for delimiters to be defined so that they can be mapped successfully from one OTD to another. The X12 delimiters are as follows:

- Data Element Separator (default is an asterisk)
- Subelement Separator/Component Element Separator (default is a colon)
- Repetition Separator (version 4020 and later) (default is a plus sign)
- Segment Terminator (default is a tilde)

Two delimiters—Repetition Separator and Subelement Separator—are explicitly specified in the interchange header segment (ISA). The other two delimiters are implicitly defined within the structure of the ISA, by their first usage. For example, after the fourth character defines the Data Element Separator, the same character is used subsequently to delimit all data elements; and after the 107th character defines the Segment Terminator, the same character is used subsequently to delimit all segments.

Because the OTD automatically detects delimiters while unmarshaling, you need not (and should not) specify delimiters for an incoming message; any delimiters that are set before unmarshaling are ignored, and the unmarshal() function picks up the delimiter being used in the ISA segment of the incoming message.

You can specify delimiters in two ways:

- You can set the Subelement Separator and Repetition Separator from the corresponding elements within the ISA segment.
- You can set the delimiters in the Java Collaboration Editor using bean nodes that are provided in the OTDs. Specific information on using bean nodes to get and set these delimiter values is provided in **Chapter 5**:
  - elementSeparator (see **getElementSeparator** on page 58)
  - subelementSeparator (see **getSubelementSeparator** on page 62)
  - repetitionSeparator (see **getRepetitionSeparator** on page 60)
  - segmentTerminator (see **getSegmentTerminator** on page 61)

If the input data is already in X12 format, you can use the "get" methods to get the delimiters from the input data. If the Collaboration is putting the data into X12 format, you can use the "set" methods to set the delimiters in the output OTD. See **"Methods for Getting and Setting" on page 27**.

## 4.3 Methods for Getting and Setting

Bean nodes automatically have **get** and **set** methods associated with them; in other words, a bean node named *theBeanNode* has a method **get*TheBeanNode*()** to read the current value and another method **set*TheBeanNode*()** to write a value. Therefore, do not assume that a node is read/write merely because it has a **set*Node*()** method.

The following bean nodes are available under the root node and at the *xxx*_Outer, *xxx*_Inner, and *xxx* (transaction set) levels:

- **elementSeparator**(char)— to get or set the element separator.
- **inputSource**(byte[])— to get the byte array of original input data source.
- **repetitionSeparator**(char)— to get or set the repetition separator.
- **segmentCount**(int)— to get the segment count at the current level. This node is also available for segment loops.
- **segmentTerminator**(char)— to get or set the segment terminator.
- **subelementSeparator**(char)— to get or set the subelement separator.
- **xmlOutput**(boolean)—to set whether the output should be in XML format.

The following bean node is available from the Loop elements:

- **segmentCount**(int)— to get the segment count at the current level. This node is also available under the root node and at the *xxx*_Outer, *xxx*_Inner, and *xxx* (transaction set) levels.

***Note:*** *Additional information on methods included in the HIPAA OTD Library are found in* **Bean Nodes and Java Methods** *on page 35.*

## 4.4 Using Validation in the Java Collaboration Editor

Each of the OTDs in the HIPAA OTD library includes a Java method for the purpose of validating your data:

- **performValidation***()*

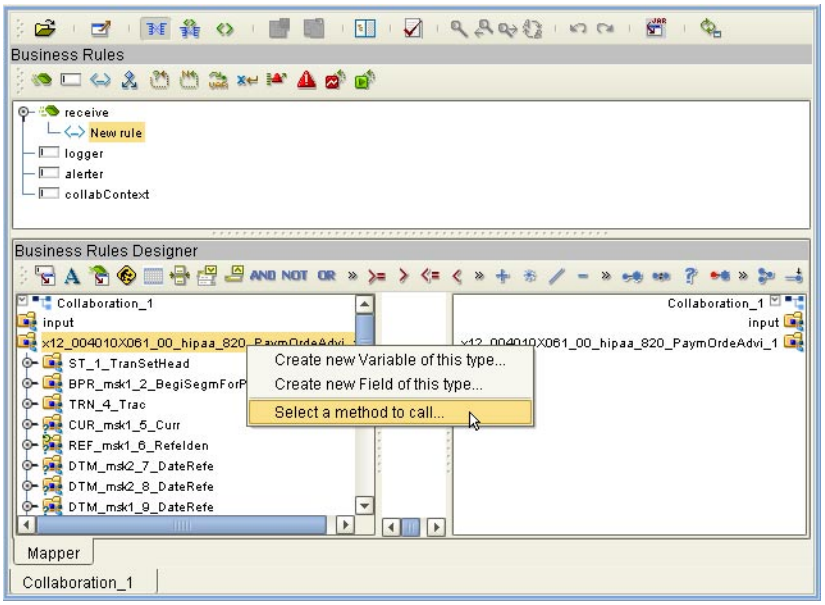
Information on using this method from within Java Collaboration Editor (JCE) GUI is provided below. Technical information on the Java methods is provided in **"Java Methods for X12 OTDs" on page 52**.

### 4.4.1. Creating a Collaboration Rule to Validate a HIPAA OTD

The elements that are part of an OTD can be dragged and dropped when two or more OTDs are opened in the Java Collaboration Editor; see the *eGate Integrator User's Guide* for more information. A field on the input (left) side pane can be dragged to a field in the output (right) pane. This action, when highlighted in the Business Rules pane, displays the rule in the Rule Properties pane.

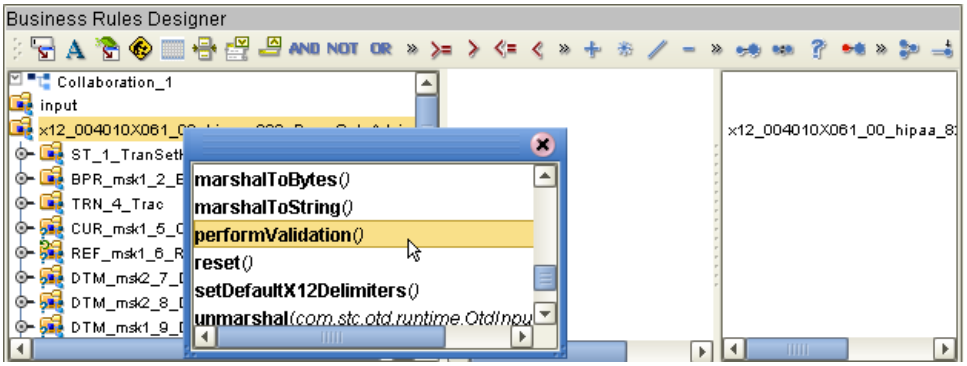
To access the method, right-click the node and, on the context popup menu, click **Select a method to call**. See Figure 6.

**Figure 6** Accessing a Method in an X12 OTD




The methods available depend on the node you select. In particular, if you right-click the root node of the OTD, one of the methods available to you is **performValidation()**; see Figure 7.

**Figure 7** Accessing the performValidation Method from the Root Node




The **performValidation()** method can be used to validate a HIPAA message at run time. If the OTD content is found to be invalid, the appropriate error bean nodes are populated (see **"Bean Nodes for Reporting Errors and Exceptions" on page 53**). Therefore, the complete set of bean nodes for reporting errors and exceptions can only be accessed after the call to **performValidation()**.

***Note:*** *Although validation is a useful tool to ensure that data conforms to the definitions and business rules, be aware that it significantly impacts performance.*

## 4.4.2. Bean Nodes for Getting Errors and Results

The following bean nodes are available under the root node and at the *xxx*_Outer, *xxx*_Inner, and *xxx* (transaction set) levels.

- **allErrors**(String[])— to get errors during unmarshaling from the input data and validation results on message and envelopes, in the format of a String array that combines (without duplication) the results from ICValidationResult(), FGValidationResult(), TSValidationResult(), and msgValidationResult().
- **ICValidationResult**(com.stc.otd.runtime.check.sef.ICError[])— to get the interchange envelope validation result, in the format of an array of com.stc.otd.runtime.check.sef.**ICError** objects.
- **FGValidationResult**(com.stc.otd.runtime.check.sef.FGError[])— to get the functional group envelope validation result in the format of an array of com.stc.otd.runtime.check.sef.**FGError** objects.
- **TSValidationResult**(com.stc.otd.runtime.check.sef.TSError[])— to get the transaction set envelope validation result in the format of an array of com.stc.otd.runtime.check.sef.**TSError** objects.
- **maxDataError**(int)— to get or set the maximum number of validation errors to be reported, where **-1** means "no limit."
- **msgValidationResult**(com.stc.otd.runtime.check.sef.DataError[])— to get validation errors, in the format of com.stc.otd.runtime.check.sef.**DataError** objects.
- **unmarshalErrors**(com.stc.otd.runtime.check.sef.DataError[])— to get errors that occurred during unmarshaling from the input data, in the format of

com.stc.otd.runtime.check.sef.**DataError** objects. The presence of any objects in this array implies that **isUnmarshalComplete()** is false.

***Note:*** *Additional information on methods included in the HIPAA OTD Library are found in* **Bean Nodes and Java Methods** *on page 35.*

# 4.5 HIPAA OTD Components Naming Conventions

Each HIPAA OTD contains envelope, transaction, segment loop, segment, and element names. In addition, there may be mask names and composite names. The components in an OTD correspond to the components in the X12 transaction type represented by the OTD. The component names are very similar to the names listed in the X12 implementation guides, with some abbreviations and additional SEF ordinal number information to help you determine which instance of a repeating component is referenced.

## Envelope and Transaction Names

Each OTD contains two envelope names and a transaction name. The transaction name always begins with X12, followed by version information, the transaction type ID, and a short description. For example, a standard OTD transaction name is **X12_004010X096_00_hipaaQ3_837_HealCareClai**. This means it is an X12 transaction, based on the May 2000 standards, and it is a HIPAA 837 Professional Health Care Claim.

The transaction name for an Addenda OTD is similar to the standard OTD transaction name, except the version number always ends with "A1". For example, an addenda OTD transaction name is **X12_004010X096A1_00_hipaaQ3_837_HealCareClai**.

The Interchange Group level is indicated by appending "Outer" to the transaction name; for example, **X12_004010X096_00_hipaaQ3_837_HealCareClaiOuter**.

The Functional Group level is indicated by appending "Inner" to the transaction name; for example, **X12_004010X096_00_hipaaQ3_837_HealCareClaiInner**.

## Segment Loop Names

Segment loop names are similar to the standard names in the X12 implementation guides, with some qualifiers. Each segment loop name begins with "Loop", followed by the name of the segment loop, the segment loop ordinal number based on SEF specifications, and a short description of the loop (that is, the first four characters of the loop). An example of a segment name would be **Loop2010AB_16_2010**, indicating loop 2010AB with an SEF ordinal number of 16.

## Segment Names

In the HIPAA X12 OTDs, each segment name begins with the segment ID, followed by a mask number if applicable, the segment ordinal number based on SEF specifications,

and a short description of the loop. For example, **N3_24_AddrInfo** indicates the address information segment, N3, with an SEF ordinal number of 24. Mask numbers are prefaced by "msk". **SBR_msk2_21_SubsInfo** is an example of a segment name with a mask number.

Segments in Interchange or Functional Group envelopes use a different formatting for the naming convention. These names begin with "GS" followed by a short description; for example, **GS_FuncGrouHead** indicates the Functional Group Header segment.

### Composite names

Composite names within the HIPAA OTDs begin with the composite ID, which is followed by the composite ordinal in the segment and a short description. For example, **C003_3_CompMediProcIden** indicates composite C003, Composite Medical Procedure Identifier, with an ordinal number of 3. Like segments, composite names can include a mask number, which appears just after the composite ID; for example, **C022_msk1_1_HealCareCodeInfo**.

### Element names

Element names within the HIPAA OTDs are indicated by the letter "E" at the beginning of the name. This is followed by the element ID, the element ordinal number in the segment, and a short description. For example, **E1138_1_PayeRespSequNumbCode** represents element 1138, Payer Responsibility Sequence Number Code, which is the first element in the segment.

## 4.6 Extending OTDs

Currently SeeBeyond does not support the editing of pre-built OTDs.

## 4.7 Alternative Formats: ANSI and XML

All the HIPAA OTDs accept either standard ANSI X12 format or XML format as input, by default; and, by default, output from a collaboration that uses messages from an X12 OTD is in ASC X12 format. However, there is a Java method available for setting the output to XML:

- setXMLOutput (boolean isXML)

If you want to set the collaboration to output XML format, use setXmlOutput(true); in other words, set the xmlOutput bean node to the value **true.**

### 4.7.1. XML Format for HIPAA X12

Since there is no established XML standard for X12 as yet, the SeeBeyond HIPAA X12 OTD Library uses Open Business Objects for EDI (OBOE) as the XML format for X12.

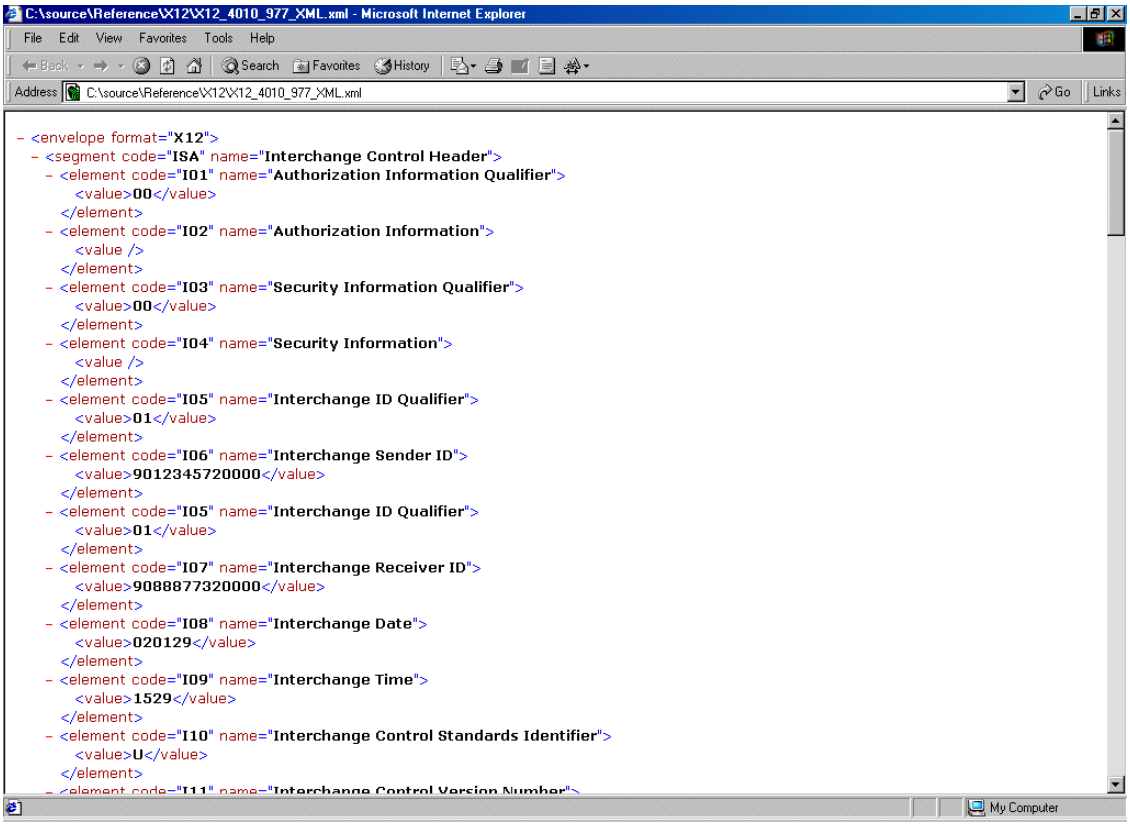
The XML X12 DTD is shown in Figure 8.

**Figure 8** XML X12 DTD

```
<!ELEMENT envelope (segment, segment?, functionalgroup+, segment)>
<!ATTLIST envelope format CDATA #IMPLIED>

<!ELEMENT functionalgroup (segment, transactionset+, segment)>

<!ELEMENT transactionset (table+)>
<!ATTLIST transactionset code CDATA #REQUIRED>
<!ATTLIST transactionset name CDATA #IMPLIED>

<!ELEMENT table (segment)+>
<!ATTLIST table section CDATA #IMPLIED>

<!ELEMENT segment ((element | composite)+, segment*)>
<!ATTLIST segment code CDATA #REQUIRED>
<!ATTLIST segment name CDATA #IMPLIED>

<!ELEMENT composite (element)+>
<!ATTLIST composite code CDATA #REQUIRED>
<!ATTLIST composite name CDATA #IMPLIED>

<!ELEMENT element (value)>
<!ATTLIST element code CDATA #REQUIRED>
<!ATTLIST element name CDATA #IMPLIED>

<!ELEMENT value (#PCDATA)>
<!ATTLIST value description CDATA #IMPLIED>
```

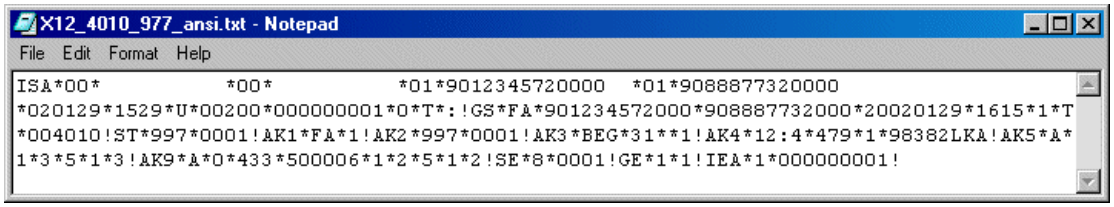
Figure 9 shows an X12 997 Functional Acknowledgment, in XML format.

**Figure 9** X12 997 Functional Acknowledgment—XML


C:\source\Reference\X12\X12_4010_977_XML.xml - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Search Favorites History
Address C:\source\Reference\X12\X12_4010_977_XML.xml
Go Links
- <envelope format="X12">
- 
- <element code="I01" name="Authorization Information Qualifier">
<value>00</value>
</element>
- <element code="I02" name="Authorization Information">
<value />
</element>
- <element code="I03" name="Security Information Qualifier">
<value>00</value>
</element>
- <element code="I04" name="Security Information">
<value />
</element>
- <element code="I05" name="Interchange ID Qualifier">
<value>01</value>
</element>
- <element code="I06" name="Interchange Sender ID">
<value>9012345720000</value>
</element>
- <element code="I05" name="Interchange ID Qualifier">
<value>01</value>
</element>
- <element code="I07" name="Interchange Receiver ID">
<value>9088877320000</value>
</element>
- <element code="I08" name="Interchange Date">
<value>020129</value>
</element>
- <element code="I09" name="Interchange Time">
<value>1529</value>
</element>
- <element code="I10" name="Interchange Control Standards Identifier">
<value>U</value>
</element>
My Computer

An example of the same transaction, an X12 997 Functional Acknowledgment, using standard ANSI format, is shown in Figure 10.

**Figure 10** X12 997 Functional Acknowledgment—ANSI Format



# 4.8 Possible Differences in Output When Using Pass-Through

If you are using pass-through, the output file contains essentially the same data as the input file.

Certain differences in output, based on variations in acceptable interpretation of the information, are acceptable, provided that the data conforms to the formats specified for the elements. For example:

- If the input file includes a six-digit date, the output file might represent this as an eight-digit value. For example, 010420 in the input file might be represented as 20010420 in the output file.
- The number of trailing zeros after a decimal point might vary. For example, an input value of 10.000 might be represented as 10 in the output file.

The reason these changes occur is that, during pass-through, certain data fields are parsed and stored as Java objects other than strings; for example, Date or Double.

The actual value of all the information must remain the same.

Chapter 5

# Bean Nodes and Java Methods

The HIPAA OTD Library contains bean nodes and Java methods that are used to extend the functionality of the OTDs. This chapter describes these methods, and includes descriptions of the output generated by the validation and error message output methods.

## 5.1 Bean Nodes

All bean nodes have get methods associated with them; in other words, a bean node named theBeanNode has a method getTheBeanNode() to read the current value.

In addition to the get methods that all bean nodes have, read/write bean nodes have set methods; that is, the method setTheBeanNode() writes a value.

These methods can be used together or separately. For example, they can allow you to get the X12 delimiters from the input OTD and set them appropriately for the output OTD; or they can also allow you to set the delimiters to the default values.

Bean nodes included in the HIPAA OTD Library can be grouped under the following categories:

- Delimiter related
- inputSource
- Validation and error reporting

### 5.1.1. Delimiter Related Bean Nodes

Delimiter related bean nodes found in the HIPAA OTD Library include:

- segmentTerminator (See **getSegmentTerminator** on page 41)
- ElementSeparator (See **getElementSeparator** on page 42)
- subelementSeparator (See **getSubelementSeparator** on page 43
- repetitionSeparator (See **getRepetitionSeparator** on page 44)

### 5.1.2. inputSource Related Bean Nodes

inputSource related bean nodes found in the HIPAA OTD Library include:

- **xmlOutput**—This bean node is of data type **boolean**. The value determines whether the marshal() method generates XML output. The default value is **false**, which causes the marshal() method to generate delimited output. Setting the value to true invokes two marshal methods—marshalToBytes() and marshalToString()—that output data in XML format.
- **SegmentCount**—This bean node is of data type **int**. It gives the number of segments at the current node. It is available at the following levels:
  - *xxx*_**Outer** (from ISA to IEA segments)
  - *xxx*_**Inner** (from GS to GE segments)
  - *xxx*_<*transactionset*> (from ST to SE segments)

### 5.1.3. Validation and Error Processing Related Bean Nodes

Validation and error processing related bean nodes found in the HIPAA OTD Library include:

- **msgValidationResult**—Data type **com.stc.otd.runtime.check.sef.DataError[]**.This array stores all exceptions that occur during the validation of the message (in other words, the segments between, but not including, the ST and SE segments).
- **unmarshalErrors**—Data type **com.stc.otd.runtime.check.sef.DataError[]**. This array stores all exceptions that occur when the unmarshal() method is invoked and while accessing elements of a segment for the first time. If this array has any objects in it before a call to **performValidation** has been made, their presence is equivalent to isUnmarshalComplete() returning false.
- **ICValidationResult**—Data type **com.stc.otd.runtime.check.sef.ICError[]**.This array stores all exceptions that occur during the validation of the Interchange envelope.
- **FGValidationResult**—Data type **com.stc.otd.runtime.check.sef.FGError[]**.This array stores all exceptions that occur during validation of the Function Group envelope (in other words, the GS and GE segments).
- **TSValidationResult**—Data type **com.stc.otd.runtime.check.sef.TSError[]**.This array stores all exceptions that occur during validation of the Transaction set (in other words, the ST and SE segments).
- **allErrors**—This bean node is of data type **java.lang.String[]**. It holds a string version of every exception stored in the other bean nodes listed below: unmarshalErrors, ICValidationResult, FGValidationResult, TSValidationResult, and msgValidationResult.
- **maxDataError**—Determines the maximum number of message validation errors to be reported into the msgValidationResult bean node. If set to -1 (default), then there is no limit to the number of errors to be reported in the msgValidationResult bean node.

# 5.2 Java Methods

In addition to the bean nodes described in the previous section, and whose methods are explained below, the top node of any x12_0040??_[...] OTD in the HIPAA OTD Library also includes the following Java methods:



## check

### Description

Performs validation on the OTD unmarshaled from inbound data, and returns message validation result into a String array.

### Parameters

None.

### Throws

None.

### Returns

**String[]** (the message validation result)

## isUnmarshalComplete

### Description

Flag for whether or not the unmarshaling (parsing) has completed successfully.

### Parameters

None.

### Throws

None.

### Returns

**boolean** (whether or not the initial explicit call to **unmarshal** completed successfully). For caveats and limitations, see **"Delayed Unmarshaling" on page 50** and **"Errors and Exceptions" on page 50**

---

## marshalToBytes

### Description

Marshals (serializes, renders) the internal data tree into a byte array.

### Parameters

in - **byte[]** (the input, as a byte array)

### Throws

**java.io.IOException** (for output problems)

**com.stc.otd.runtime.MarshalException** (for an inconsistent internal tree)

### Returns

**byte[]** (serialized byte array)

---

## marshalToString

### Description

Marshals (serializes, renders) the internal data tree into a String.

### Parameters

None.

### Throws

**java.io.IOException** (for input problems)

**com.stc.otd.runtime.MarshalException** (for an inconsistent internal tree)

### Returns

**java.lang.String** (the serialized String)

---

## performValidation

### Description

Validates the OTD content immediately after unmarshaling.

### Syntax

```
public void performValidation()
```

### Parameters

None.

### Constants

None.

### Returns

None. If the OTD content is found to be invalid, the appropriate error bean nodes are populated.

### Throws

None.

### Examples

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
myOTD.performValidation();
```

### Notes

For an example of using **performValidation()** in a collaboration, see **Creating a Collaboration Rule to Validate an X12 OTD** on page 45.

---

## reset

### Description

Clears out any data and resources held by this OTD instance.

### Parameters

None.

### Throws

None.

### Returns

None.

---

## setDefaultX12Delimiters

### Description

Sets the default X12 delimiters, such as:

| | |
|---|---|
| **~** | segment terminator |
| ***** | element separator |
| **:** | subelement separator |
| **+** | repetition separator |

### Syntax

```
public void setDefaultX12Delimiters()
```

### Parameters

None.

### Constants

None.

### Returns

void (none).

### Throws

None.

### Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
myOTD.setDefaultX12Delimiters();
```

## unmarshalFromBytes

### Description

Unmarshals (deserializes, parses) the given input byte array into an internal data tree.

### Parameters

in - **byte[]** (the input, as a byte array)

### Throws

**java.io.IOException** (for input problems)

**com.stc.otd.runtime.UnmarshalException** (for an inconsistent internal tree)

### Returns

None.

## unmarshalFromString

### Description

Unmarshals (deserializes, parses) the given input string into an internal data tree.

### Parameters

in - **java.lang.String** (the input, as a String)

### Throws

**java.io.IOException** (for input problems)

**com.stc.otd.runtime.UnmarshalException** (for an inconsistent internal tree; typically occurs if the OTD cannot recognize the incoming message as X12)

### Returns

None.

## getSegmentTerminator

### Description

Gets the segmentTerminator character.

### Syntax

```
public char getSegmentTerminator()
```

### Parameters

None.

### Constants

None.

### Returns

**char**
Returns the segment terminator character.

### Throws

None.

### Examples

```
com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter
 input=new com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_Heal
 CareClaiOuter();
......
......
char segTerm=input.getSegmentTerminator();
```

## setSegmentTerminator

### Description

Sets the segmentTerminator character.

### Syntax

```
public void setSegmentTerminator(char c)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | The character to be set as the segment terminator. |

### Constants

None.

### Returns

void (none).

### Throws

None.

### Examples

```
com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter
 input=new com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_Heal
 CareClaiOuter();
......
......
char c='~';
input.setSegmentTerminator(c);
```

## getElementSeparator

### Description

Gets the elementSeparator character.

### Syntax

```
public char getElementSeparator()
```

### Parameters

None.

### Constants

None.

### Returns

**char**
Returns the element separator character.

### Throws

None.

### Examples

```
com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter
 input=new com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_Heal
 CareClaiOuter();
......
......
char elmSep=input.getElementSeparator();
```

## setElementSeparator

### Description

Sets the elementSeparator character.

### Syntax

```
public void setElementSeparator(char c);
```

### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | The character to be set as the element separator. |

### Constants

None.

### Returns

void (none).

### Throws

None.

### Examples

```
com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter
 input=new com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_Heal
 CareClaiOuter();
......
......
char c='+';
input.setElementSeparator(c);
```

## getSubelementSeparator

### Description

Gets the subelementSeparator character.

### Syntax

```
public char getSubelementSeparator()
```

### Parameters

None.

### Constants

None.

### Returns

**char**

Returns the subelement separator character.

### Throws

None.

### Examples

```
com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter
 input=new com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_Heal
 CareClaiOuter();
......
......
char subeleSep=input.getSubelementSeparator();
```

## setSubelementSeparator

### Description

Sets the subelementSeparator character.

### Syntax

```
public void setSubelementSeparator(char c)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | The character to be set as the subelement separator. |

### Constants

None.

### Returns

void (none).

### Throws

None.

### Examples

```
com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter
 input=new com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_Heal
 CareClaiOuter();
......
......
char c=':';
input.setSubelementSeparator(c);
```

## getRepetitionSeparator

### Description

Gets the repetitionSeparator character.

### Syntax

```
public char getRepetitionSeparator()
```

### Parameters

None.

### Constants

None.

### Returns

**char**
Returns the repetition separator character.

### Throws

None.

### Examples

```
com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter
 input=new com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_Heal
 CareClaiOuter();
......
......
char repSep=input.getRepetitionSeparator();
```

## setRepetitionSeparator

### Description

Sets the repetitionSeparator character.

### Syntax

```
public void setRepetitionSeparator(char c)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| c | char | The character to be set as the repetition separator. |

### Constants

None.

### Returns

void (none).

### Throws

None.

### Examples

```
com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter
 input=new com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_Heal
 CareClaiOuter();
......
......
char c='*';
input.setRepetitionSeparator(c);
```

## getUnmarshalErrors

### Description

Retrieves an array of unmarshal error objects of the type "DataError". Call this after **isUnmarshalComplete** returns false, which indicates that unmarshalling was not finished due to errors. As an alternative, you can call **getMsgValidationResult** or **getAllErrors**.

***Note:*** *When **getMsgValidationResult** or **getAllErrors** is called without first calling **performValidation**, they return an array of unmarshalling errors.*

### Syntax

```
public com.stc.hipaa.DataError[] getUnmarshalErrors()
```

### Parameters

None.

### Constants

None.

### Returns

**com.stc.hipaa.DataError[]**
An array of errors found during the validation process.

### Throws

None.

### Examples

```
com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter
 input=new com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_Heal
 CareClaiOuter();
com.stc.hipaa.X12ValidationResult output=new
com.stc.hipaa.X12ValidationResult();
.....
if (input().isUnmarshalComplete())
.....
else
   output().addDataErrors(input().getUnmarshalErrors());
```

## getMsgValidationResult

### Description

Returns an array of errors found during unmarshalling from the input data and any errors found during validation (**performValidation**).

### Syntax

```
public com.stc.hipaa.DataError[] getMsgValidationResult()
```

### Parameters

None.

### Constants

None.

### Returns

**com.stc.hipaa.DataError[]**
An array of errors found in the input data during unmarshalling and the validation process.

### Throws

None.

### Examples

```
com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter
 input=new com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_Heal
 CareClaiOuter();
com.stc.hipaa.X12ValidationResult output=new
com.stc.hipaa.X12ValidationResult();
.....
.....
if (input().isUnmarshalComplete()) {
   input().performValidation();
   output().addDataErrors(input().getMsgValidationResult());
```

## getAllErrors

### Description

Outputs an array of string representations of all errors that occurred during unmarshalling from the input data and the validation results for both the message and envelopes.

***Note:*** *To view a sample error message that getAllErrors would output, see "Validation Error Reporting" in Chapter 4 of the HIPAA Implementation Guide. This section provides information about each element in the error code.*

### Syntax

```
public java.lang.String[] getAllErrors()
```

### Parameters

None.

### Constants

None.

### Returns

**String[]**
A string array of error messages describing any errors in the input data. If there are no errors, the array size is zero (0).

### Throws

None.

### Examples

```
com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter
 input=new com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_Heal
 CareClaiOuter();
com.stc.hipaa.X12ValidationResult output=new
com.stc.hipaa.X12ValidationResult();
.....
.....
if (input().isUnmarshalComplete()) {
   input().performValidation();
   output().addDataErrors(input().getAllErrors());
```

## getICValidationResult

### Description

Outputs results from **performValidation**, but only outputs results of the interchange (IC) envelope validation.

***Note:*** *Only certain IC validations are performed. For more information about IC validations, see* **"getICValidationResult" on page 48***.*

### Syntax

```
public com.stc.hipaa.ICError[] getICValidationResult()
```

### Parameters

None.

### Constants

None.

### Returns

**com.stc.hipaa.ICError[]**
An array of interchange envelope errors found during the validation process.

### Throws

None.

### Examples

```
com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter
 input=new com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_Heal
 CareClaiOuter();
com.stc.hipaa.X12ValidationResult output=new
com.stc.hipaa.X12ValidationResult();
.....
.....
if (input().isUnmarshalComplete())
   input().performValidation();
   output().addDataErrors(input().getICValidationResult());
```

## getFGValidationResult

### Description

Outputs the results of **performValidation**, but only outputs results of the functional group (FG) envelope validation.

***Note:*** *Only certain FG validations are performed. For more information about FG validations, see* **"getICValidationResult" on page 48***.*

### Syntax

```
public com.stc.hipaa.FGError[] getFGValidationResult()
```

### Parameters

None.

### Constants

None.

### Returns

**com.stc.hipaa.FGError[]**
An array of the functional group envelope errors found during the validation process.

### Throws

None.

### Examples

```
com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter
 input=new com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_Heal
 CareClaiOuter();
com.stc.hipaa.X12ValidationResult output=new
com.stc.hipaa.X12ValidationResult();
.....
.....
if (input().isUnmarshalComplete())
   input().performValidation();
   output().addDataErrors(input().getFGValidationResult());
```

## getTSValidationResult

### Description

Outputs the result of **performValidation**, but only outputs results of the transaction/message (TS) envelope validation.

***Note:*** *Only certain TS validations are performed. For more information about TS validations, see* **"getICValidationResult" on page 48***.*

### Syntax

```
public com.stc.hipaa.TSError[] getTSValidationResult()
```

### Parameters

None.

### Constants

None.

### Returns

**com.stc.hipaa.TSError[]**
An array of transaction/message envelope errors found during the validation process.

### Throws

None.

### Examples

```
com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter
 input=new com.stc.hipaa837_Q120.X12_004010X098_00_hipaaQ1_837_Heal
 CareClaiOuter();
com.stc.hipaa.X12ValidationResult output=new
com.stc.hipaa.X12ValidationResult();
.....
.....
if (input().isUnmarshalComplete())
   input().performValidation();
   output().addDataErrors(input().getTSValidationResult());
```

Appendix A

# ASC X12 Overview

This appendix provides an overview of the X12 standard, including:

- An overview of X12, including the structure of an X12 envelope, data elements, and syntax.
- An explanation of how to use the generic message structures provided as an add-on to the eGate to help you quickly create the structures you need for various X12 transactions.

For specific information on HIPAA, refer to **Chapter 2**, **"HIPAA Overview" on page 11**.

## A.1 Introduction to X12

The following sections provide an introduction to X12.

### A.1.1. What Is ASC X12?

ASC X12 is an EDI (electronic data interchange) standard, developed for the electronic exchange of machine-readable information between businesses.

The Accredited Standards Committee (ASC) X12 was chartered by the American National Standards Institute (ANSI) in 1979 to develop uniform standards for interindustry electronic interchange of business transactions—electronic data interchange (EDI). The result was the X12 standard.

The ASC X12 body develops, maintains, interprets, and promotes the proper use of the ASC X12 standard. Data Interchange Standards Association (DISA) publishes the ASC X12 standard and the UN/EDIFACT standard. The ASC X12 body comes together three times a year to develop and maintain EDI standards. Its main objective is to develop standards to facilitate electronic interchange relating to business transactions such as order placement and processing, shipping and receiving information, invoicing, and payment information.

The ASC X12 EDI standard is used for EDI within the United States. UN/EDIFACT is broadly used in Europe and other parts of the world.

X12 was originally intended to handle large batches of transactions. However, it has been extended to encompass real-time processing (transactions sent individually as they are ready to send, rather than held for batching) for some healthcare transactions to accommodate the healthcare industry.

### A.1.2. What Is a Message Structure?

The term *message structure* (also called a transaction set structure) refers to the way in which data elements are organized and related to each other for a particular EDI transaction.

In eGate, a message structure is called an Object Type Definition (OTD). Each message structure (OTD) consists of the following:

- Physical hierarchy

  The predefined way in which envelopes, segments, and data elements are organized to describe a particular X12 EDI transaction.

- Delimiters

  The specific predefined characters that are used to mark the beginning and end of envelopes, segments, and data elements.

- Properties

  The characteristics of a data element, such as the length of each element, default values, and indicators that specify attributes of a data element—for example, whether it is required, optional, or repeating.

The transaction set structure of a claim that is sent from a payer to a provider defines the header, trailer, segments, and data elements required by claim transactions. Installation of HIPAA OTDs for a specific version includes transaction set structures for each of the transactions available in that version.

The HIPAA OTD Library provides eGate Object Type Definitions, which are based on the X12 message structures, to verify that the data in the messages coming in or going out is in the correct format. There is a message structure for each transaction.

The list of transactions provided is different for each version of X12, and for each customized implementation. This book addresses the transactions covered by the May 1999 and May 2000 implementations of the HIPAA standard.

## A.2 Components of an X12 Envelope

X12 messages are all ASCII text, with the exception of the BIN segment which is binary.

Each X12 message is made up of a combination of the following elements:

- Data elements
- Segments
- Loops

Elements are separated by delimiters.

More information on each of these is provided below.

## A.2.1. Data Elements

The data element is the smallest named unit of information in the ASC X12 standard. Data elements can be broken down into two types. The distinction between the two is strictly a matter of how they are used. The two types are:

- Simple

  If a data element occurs in a segment outside the defined boundaries of a composite data structure it is called a simple data element.

- Composite

  If a data element occurs as an ordinally positioned member of a composite data structure it is called a composite data element.

Each data element has a unique reference number; it also has a name, description, data type, and minimum and maximum length.

## A.2.2. Segments

A segment is a logical grouping of data elements. In X12, the same segment can be used for different purposes. This means that a field's meaning can change based on the segment. For example:

- The NM1 segment is for *any* name (patient, provider, organization, doctor)
- The DTP segment is for *any* date (date of birth, discharge date, coverage period)

For more information on the X12 enveloping segments, refer to **"Structure of an X12 Envelope" on page 54**.

## A.2.3. Loops

Loops are sets of repeating ordered segments. In X12 you can locate elements by specifying:

- The transaction set (for example, 270)
- The loop (for example, "loop 1000" or "info. receiver loop")
- The occurrence of the loop
- The segment (for example, BGN)
- The field number (for example, 01)
- The occurrence of the segment (if it is a repeating segment)

## A.2.4. Delimiters

In an X12 message, the various delimiters act as syntax, dividing up the different elements of a message. The delimiters used in the message are defined in the interchange control header, the outermost layer enveloping the message. For this reason, there is flexibility in the delimiters that are used.

No suggested delimiters are recommended as part of the X12 standards, but the industry-specific implementation guides do have recommended delimiters.

The default delimiters used by the SeeBeyond HIPAA OTD Library are the same as those recommended by the industry-specific implementation guides. These delimiters are shown in Table 6.

**Table 6** Default Delimiters in X12 OTD Library

| Type of Delimiter | Default Value |
|---|---|
| Segment terminator | ~ (tilde) |
| Data element separator | * (asterisk) |
| Subelement (component) separator | : (colon) |

***Note:*** *It is important to note that errors could result if the transmitted data itself includes any of the characters that have been defined as delimiters. Specifically, the existence of asterisks within transmitted application data is a known issue in X12, and can cause problems with translation.*

## A.3 Structure of an X12 Envelope

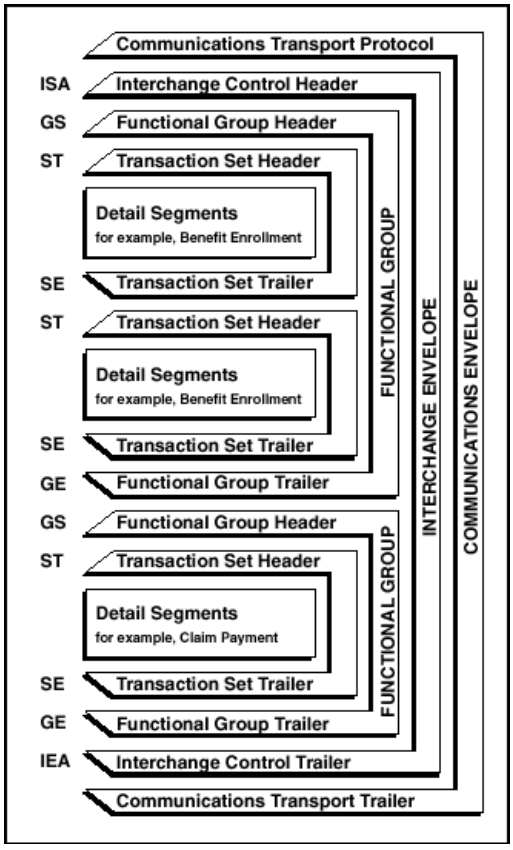
The rules applying to the structure of an X12 envelope are very strict to ensure the integrity of the data and the efficiency of the information exchange.

The actual X12 message structure has three main levels. From the highest to the lowest they are:

- Interchange Envelope
- Functional Group
- Transaction Set

A schematic of X12 envelopes is shown in Figure 11. Each of these levels is explained in more detail in the following sections.

**Figure 11** X12 Envelope Schematic




***Note:*** *The above schematic is from Appendix B of an ASC X12 implementation guide.*

Figure 12 shows the standard segment table for an X12 997 (Functional Acknowledgment) as it appears in the X12 standard and in most industry-specific implementation guides.

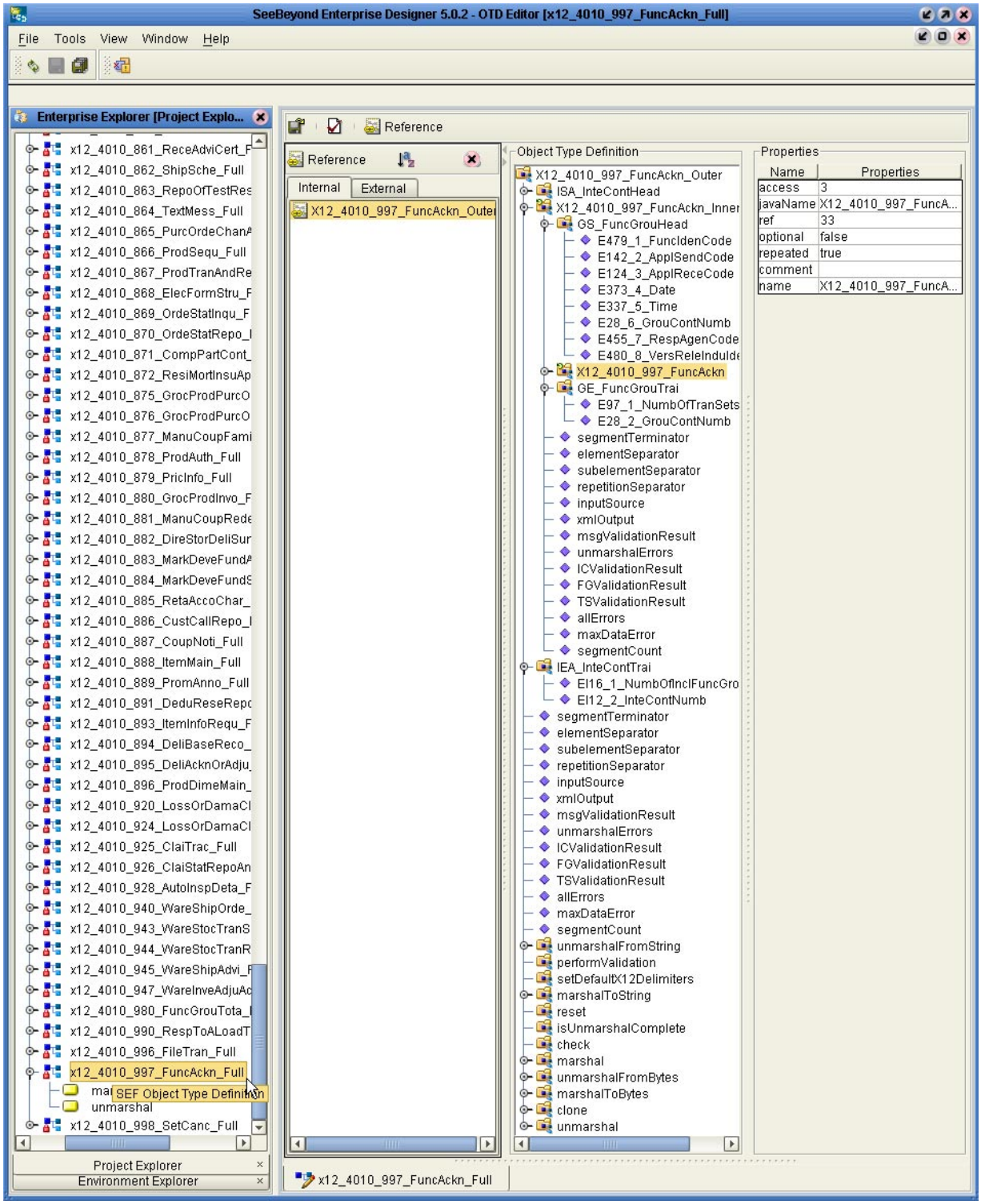
**Figure 12** X12 997 Segment Table

**Table 1 - Header**

| POS. # | SEG. ID | NAME | REQ. DES. | MAX USE | LOOP REPEAT |
|---|---|---|---|---|---|
| 010 | ST | Transaction Set Header | M | 1 | |
| 020 | AK1 | Functional Group Response Header | M | 1 | |
| | | LOOP ID - AK2 | | | 999999 |
| 030 | AK2 | Transaction Set Response Header | O | 1 | |
| | | LOOP ID - AK2/AK3 | | | 999999 |
| 040 | AK3 | Data Segment Note | O | 1 | |
| 050 | AK4 | Data Element Note | O | 99 | |
| 060 | AK5 | Transaction Set Response Trailer | M | 1 | |
| 070 | AK9 | Functional Group Response Trailer | M | 1 | |
| 080 | SE | Transaction Set Trailer | M | 1 | |

Figure 13 shows the same transaction as viewed in the OTD Editor.

**Figure 13** X12 997 (Functional Acknowledgment) Viewed in OTD Editor



## A.3.1. Transaction Set (ST/SE)

Each transaction set (also called a transaction) contains three things:

- A transaction set header
- A transaction set trailer
- A single message, enveloped within the header and footer

The transaction has a three-digit code, a text title, and a two-letter code; for example, **997, Functional Acknowledgment (FA)**.

The transaction consists of logically related pieces of information, grouped into units called segments. For example, one segment used in the transaction set might convey the address: city, state, ZIP code, and other geographical information. A transaction set can contain multiple segments. For example, the address segment could be used repeatedly to convey multiple sets of address information.

The X12 standard defines the sequence of segments in the transaction set and also the sequence of elements within each segment. The relationship between segments and elements could be compared to the relationship between records and fields in a database environment.

**Figure 14** Example of a Transaction Set Header (ST)

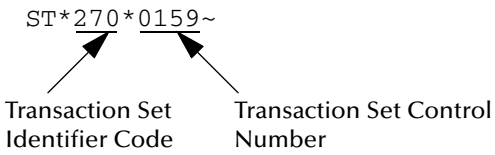
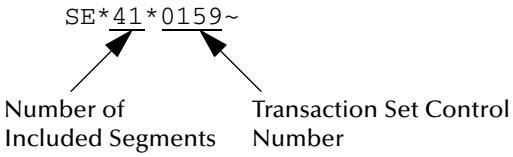



**Figure 15** Example of a Transaction Set Trailer (SE)




## A.3.2. Functional Group (GS/GE)

A functional group is comprised of one or more transaction sets, all of the same type, that can be batched together in one transmission. The functional group is defined by the header and trailer; the Functional Group Header (GS) appears at the beginning, and the Functional Group Trailer (GE) appears at the end. Many transaction sets can be included in the functional group, but all transactions must be of the same type.

Within the functional group, each transaction set is assigned a functional identifier code, which is the first data element of the header segment. The transaction sets that comprise a specific functional group are identified by this functional ID code.

The functional group header (GS) segment contains the following information:

- Functional ID code (the two-letter transaction code; for example, PO for an 850 Purchase Order, HS for a 270 Eligibility, Coverage, or Benefit Inquiry) to indicate the type of transaction in the functional group
- Identification of sender and receiver
- Control information (the functional group control numbers in the header and trailer segments must be identical)
- Date and time

The functional group trailer (GE) segment contains the following information:

- Number of transaction sets included
- Group control number (originated and maintained by the sender)

**Figure 16** Example of a Functional Group Header (GS)

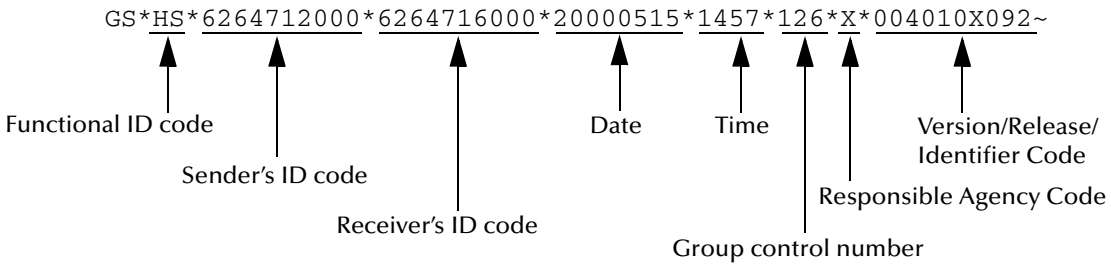
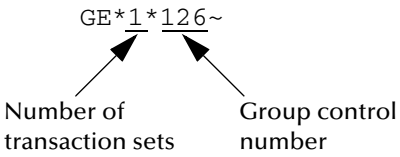



**Figure 17** Example of a Functional Group Trailer (GE)




## A.3.3. Interchange Envelope (ISA/IEA)

The interchange envelope is the wrapper for all the data to be sent in one batch. It can contain multiple functional groups. This means that transactions of different types can be included in the interchange envelope, with each type of transaction stored in a separate functional group.

The interchange envelope is defined by the header and trailer; the Interchange Control Header (ISA) appears at the beginning, and the Interchange Control Trailer (IEA) appears at the end.

As well as enveloping one or more functional groups, the interchange header and trailer segments include the following information:

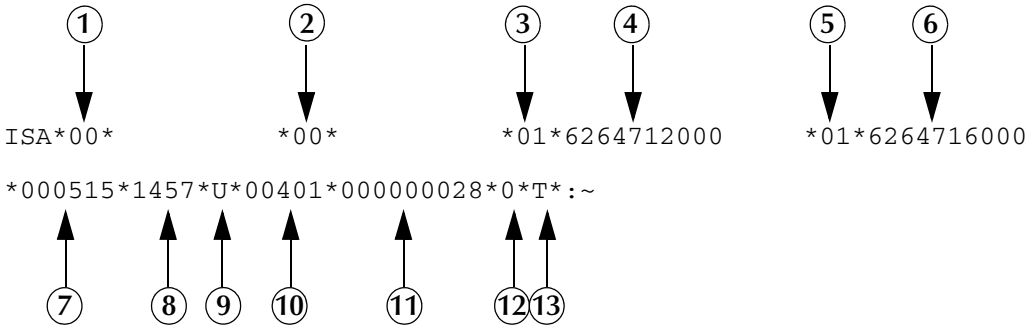
- Data element separators and data segment terminator

- Identification of sender and receiver
- Control information (used to verify that the message was correctly received)
- Authorization and security information, if applicable

The sequence of information that is transmitted is as follows:

- Interchange header
- Optional interchange-related control segments
- Actual message information, grouped by transaction type into functional groups
- Interchange trailer

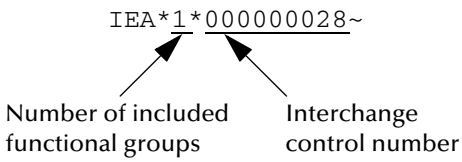
**Figure 18** Example of an Interchange Header (ISA)




Interchange Header Segments from Figure 18:

1 Authorization Information Qualifier
2 Security Information Qualifier
3 Interchange ID Qualifier
4 Interchange Sender ID
5 Interchange ID Qualifier
6 Interchange Receiver ID
7 Date
8 Time
9 Repetition Separator
10 Interchange Control Version Number
11 Interchange Control Number
12 Acknowledgment Requested
13 Usage Indicator

**Figure 19** Example of an Interchange Trailer (IEA)




## A.3.4. Control Numbers

The X12 standard includes a control number for each enveloping layer:

- ISA13—Interchange Control Number
- GS06—Functional Group Control Number
- ST02—Transaction Set Control Number

The control numbers act as identifiers, useful in message identification and tracking.

### ISA13 (Interchange Control Number)

The ISA13 is assigned by the message sender. It must be unique for each interchange.

### GS06 (Functional Group Control Number)

The GS06 is assigned by the sender. It must be unique within the Functional Group assigned by the originator for a transaction set.

***Note:*** *The Functional Group control number GS06 in the header must be identical to the same data element in the associated Functional Group trailer, GE02.*

### ST02 (Transaction Set Control Number)

The ST02 is assigned by the sender, and is stored in the transaction set header. It must be unique within the Functional Group.

***Note:*** *The control number in ST02 must be identical with the SE02 element in the transaction set trailer, and must be unique within a Functional Group (GS-GE).*

## A.4 Acknowledgment Types

X12 includes two types of acknowledgment, the TA1 Interchange Acknowledgment and the 997 Functional Acknowledgment.

### A.4.1. TA1, Interchange Acknowledgment

The TA1 acknowledgment verifies the interchange envelopes only. The TA1 is a single segment and is unique in the sense that this single segment is transmitted without the GS/GE envelope structures. A TA1 acknowledgment can be included in an interchange with other functional groups and transactions.

### A.4.2. 997, Functional Acknowledgment

The 997 includes much more information than the TA1. The 997 was designed to allow trading partners to establish a comprehensive control function as part of the business exchange process.

There is a one-to-one correspondence between a 997 and a functional group. Segments within the 997 identify whether the functional group was accepted or rejected. Data elements that are incorrect can also be identified.

Many EDI implementations have incorporated the acknowledgment process into all of their electronic communications. Typically, the 997 is used as a functional acknowledgment to a functional group that was transmitted previously.

The 997 is the acknowledgment transaction recommended by ASC X12.

The acknowledgment of the receipt of a payment order is an important issue. Most corporate originators want to receive at least a Functional Acknowledgment (997) from the beneficiary of the payment. The 997 is created using the data about the identity and address of the originator found in the ISA and/or GS segments.

Some users argue that the 997 should be used only as a point-to-point acknowledgment and that another transaction set, such as the Application Advice (824) should be used as the end-to-end acknowledgment.

### A.4.3. Application Acknowledgments

Application acknowledgments are responses sent from the destination system back to the originating system, acknowledging that the transaction has been successfully or unsuccessfully completed. The application advice (824) is a generic application acknowledgment that can be used in response to any X12 transaction. However, it has to be set up as a response transaction; only TA1 and 997 transactions are sent out automatically.

Other types of responses from the destination system to the originating system, which may also be considered application acknowledgments, are responses to query transactions—for example, the Eligibility Response (271) which is a response to the Eligibility Inquiry (270). Other types of responses from the destination system to the originating system, which may also be considered application acknowledgments, are responses to query transactions—for example, the Eligibility Response (271) which is a response to the Eligibility Inquiry (270).

## A.5 Key Parts of EDI Processing Logic

The five key parts of EDI processing logic are listed in Table 7.

**Table 7** Key Parts of EDI Processing

| Term | Description | Language Analogy | eGate Component |
|---|---|---|---|
| structures | format, segments, loops | syntax rules | OTD elements and fields |
| validations | data contents “edit” rules | semantic rules | validation methods |
| translations (also called mappings) | reformatting or conversion | translation | collaborations |
| enveloping | header and trailer segments | envelope for a written letter | the special “envelope” OTDs: FunctionalGroupEnv and InterchangeEnv |
| acks | acknowledgments | return receipt | specific acknowledgment elements in the OTD |

eGate uses the structures, validations, translations, enveloping, and acknowledgments listed below to support HIPAA.

### A.5.1. Structures

The Object Type Definition library for HIPAA includes pre-built OTDs for all supported HIPAA versions.

### A.5.2. Trading Partner Agreements

There are three levels of information that guide the final format of a specific transaction. These three levels are:

- The ASC X12 standard

  ASC X12 publishes a standard structure for each X12 transaction.

- Industry-specific Implementation Guides

  Specific industries publish Implementation Guides customized for that industry. Normally, these are provided as recommendations only. However, in certain cases, it is extremely important to follow these guidelines. Specifically, since HIPAA regulations are law, it is important to follow the guidelines for these transactions closely.

- Trading Partner Agreements

  It is normal for trading partners to have individual agreements that supplement the standard guides. The specific processing of the transactions in each trading partner's individual system might vary between sites. Because of this, additional documentation that provides information about the differences is helpful to the site's trading partners and simplifies implementation. For example, while a certain code might be valid in an implementation guide, a specific trading partner might not use that code in transactions. It would be important to include that information in a trading partner agreement.

## A.6 Additional Information

For more information on the X12 standard, visit the following Web sites:

**http://www.disa.org** and specifically **http://www.x12.org/x12org/index.cfm**

X12 implementation guides can be obtained from Washington Publishing Company:

**http://www.wpc-edi.com**; specifically, **http://www.wpc-edi.com/tg4/tg4home.asp**

***Note:*** *This information is correct at the time of going to press; however, SeeBeyond has no control over these sites. If you find the links are no longer correct, use a search engine to search for* ***X12****.*

# Index

## H



## I



## J



## L



## M



## N



## O



## P



## R



## S

## T



## U



## V



## W



## X