# HIPAA ETD Library User's Guide

*Release 5.0.5 for Schema Run-time Environment (SRE)*

Version 20100720195100.

# Contents

## Chapter 4



## Chapter 5

Chapter 1

# Introduction

This chapter provides an introduction to the HIPAA ETD 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 healthcare industry. For transactions related to healthcare, 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 the HIPAA ETD Library.

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

## 1.1 Overview

Each of the e*Gate™ Event Type Definition (ETD) libraries contains sets of pre-built structures for industry-standard formats. e*Gate ETD files are message format definitions in Monk or Java. The HIPAA ETD Library ETDs are written in Java.

The HIPAA ETD library is a feature of the Oracle™ 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 e*Gate Integrator, and provides information on installing and using the HIPAA ETD libraries.

## 1.2 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate system or the Oracle eBusiness Integration Suite, to be thoroughly familiar with Windows operations and administration, and to be familiar with Microsoft Windows graphical user interfaces.

# 1.3 Supported Operating Systems

For information about supported operating systems, see the **readme.txt** file provided on the e*Gate Integrator installation CD.

***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.*

Chapter 2

# HIPAA Overview

This chapter provides an overview of HIPAA, including general information, a list of the specific transactions that comprise the HIPAA standard, and 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 ETD 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 defines 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 had to implement these standardized transactions by October 2002 unless they obtained an extension to October 2003.

More transactions will be added for later implementations.

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 healthcare 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 ETD Library provides Event Type Definitions for the standard X12 transactions that have been adopted by HIPAA, as listed in Table 1.

The HIPAA 1999 and 2000 libraries are based on the X12 Version 4, Release 1, Sub-release 0 (004010) standard. The HIPPA 2005 library is based on the Version 5, Release1, Sub-release 0 (005010) standard.

**Table 1** 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 HIPAA 2005 library includes the additional X12 transactions listed in the following table.

**Table 2** Additional HIPAA X12 Transactions (005010)

| Number | Name |
|---|---|
| 269 | Health Care Benefit Coordination Verification |
| 275 | Patient Information Transaction Set |
| 278 | Multiple versions: Health Care Services Review and Response, Health Care Services Notification and Acknowledgement, Health Care Services Review Inquiry/Response |
| 824 | Application Advice |

The NCPDP portion of the HIPAA ETD Library provides request and response transactions for all the HIPAA-approved NCPDP transaction codes, as listed in Table 2.

**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 |
| 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 ETD Library includes both X12 and NCPDP ETDs, this document primarily discusses the HIPAA X12 ETDs. For more information about the NCPDP-HIPAA ETD Library, see the NCPDP ETD 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.

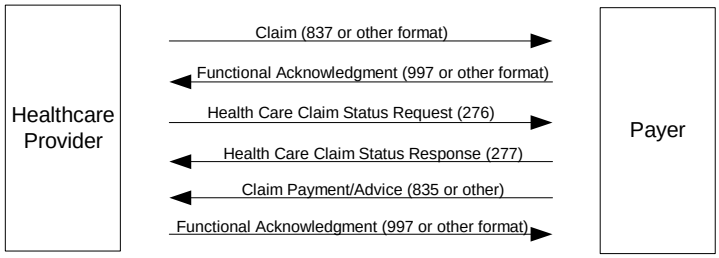
Following the HIPAA standard ensures that transactions comply with the regulations mandated by the government. HIPAA requirements are completely described in the HIPAA implementation guide for each transaction, and must not be modified by a trading partner.

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 healthcare 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 62**.

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.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, we have 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 Template Installation

This chapter provides information on the installation procedure for the HIPAA ETD library template files and shows the resulting directory structure for the templates. It includes general installation information and installation instructions.

The HIPAA ETD Library includes Java templates for the following:

- HIPAA May 1999 transactions
- HIPAA May 2000 transactions
- HIPAA 2005 (005010) transactions
- NCPDP Telecom 5.0.1/Batch 1.0 and 1.1 transactions

***Note:*** *Batch 1.1 has become the adopted standard for usage with Telecom 5.0.1. Batch 1.0 files are only provided for backward compatibility.*

Some additional points to note about the HIPAA transactions:

- The ETDs only accept messages with all the envelope segment information.
- The ETDs are intended for use with the HIPAA Collaboration Rules provided with the e*Xchange™ Partner Manager.
- Messages can be batched; however, all the messages in one functional group must be of the same message type.
- The May 1999 transaction files are only included for backward compatibility. For full HIPAA functionality, you need to use the May 2000 transaction files or later.
- Apart from their use by e*Xchange, ETDs can also be used independently for e*Gate schemas not associated with e*Xchange.

## 3.1 HIPAA Libraries

Together, ETDs and Collaboration Rules provide comprehensive validation of all eleven standard HIPAA X12 transactions. Alternately, a software patch that enables validation through a third-party network appliance is also available.

When the HIPAA ETD Library is installed, there is a separate subdirectory for each set of transactions (three in total), including both HIPAA X12 transactions (for the May 1999 and May 2000 standard) and NCPDP transactions. February 2003 amendments to

the May 2003 standard (the 00401010A1 Addenda) are stored in the May 2000 subdirectory. Each subdirectory stores all the files for that version.

For more information on the folder structure for the e*Gate HIPAA ETD Library, refer to **"HIPAA Folder Structure Created by Installation" on page 15**.

## 3.2 Installation Procedure

This section explains how to install the HIPAA template files.

**Before you begin:**

- You must have Administrator privileges to install back-end components such as the HIPAA templates.
- Exit all Windows programs, including any anti-virus applications.
- Verify your e*Gate Registry Host name, schema name, Control Broker logical name, and the administrator user name and password.

**To install the HIPAA templates on Windows**

1. Log on to the workstation on which you want to install the templates.
2. Insert the installation CD into the CD-ROM drive.

   If Autorun is enabled, the setup program automatically starts. Otherwise:

   - On the task bar, click the **Start** button, and then click **Run**.
   - In the **Open** field, type **D:\setup\setup.exe** where **D:** is your CD-ROM drive.
3. Follow the installation instructions until the **Please choose the product to install** dialog box appears.
4. Select **e*Gate Integrator**, and then click **Next**.
5. Follow the on-screen instructions until the second **Please choose the product to install** dialog box appears.
6. Select **Add-ons**, and then click **Next**.
7. Follow the on-screen instructions until the **Select Components** dialog box appears.
8. Highlight (but do not check) **ETD Libraries**, and then click the **Change** button.

   The **Select Sub-components** dialog box appears.
9. Select **Java HIPAA ETD Library**
10. Click **Continue** to return to the **Select Components** dialog box, and then click **Next**.
11. Follow the rest of the on-screen instructions to install the HIPAA templates.

    For more information about installing e*Gate add ons, see the *e*Gate Integrator Installation Guide*.

***Note:*** *Do not change the default directory location for the HIPAA template files.*

**To install the HIPAA templates on UNIX**

1. Follow the steps for the standard e*Gate installation.

   For more information, refer to the *e*Gate Integrator Installation Guide*.

2. At the prompt **Choose e*Gate Add-on Application**, enter the number assigned to the Java HIPAA Library (scroll down the list to check the number).
3. Enter the installation path, or press **Enter** to accept the default path (recommended).
4. Enter the hostname of the Registry server (UNIX host).

   The library is installed.

***Important:*** *In order for the complete e*Xchange HIPAA solution to work properly, you must also install the X12 4010 ETD Library. This ensures proper message envelope validation. See the X12 ETD Library User's Guide for more information.*

## 3.3 HIPAA Files and Folders

This section outlines the folder structure created on your hard drive as a result of installation of the HIPAA templates, and the files copied into those folders. The files include Event 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. HIPAA Folder Structure Created by Installation

By default, installation places the HIPAA ETD templates in the locations shown in Table 3.

**Table 3** HIPAA Template Locations

| These files… | Are installed in this location… |
|---|---|
| 1999 | \<eGate>\Server\Registry\Repository\default\ETD\templates\HIPAA_1999 |
| 2000 | \<eGate>\Server\Registry\Repository\default\ETD\templates\HIPAA_2000 |
| Addenda | \<eGate>\Server\Registry\Repository\default\ETD\templates\HIPAA_2000 |
| 2005 | \<eGate>\Server\Registry\Repository\default\ETD\templates\HIPAA_2005 |

***Note:*** *The e*Xchange schema includes a pair of Monk HIPAA ETDs for backward compatibility. One Monk ETD is provided for NCPDP Batch 1.1 transactions, and one is provided for Batch 1.0 transactions. These ETDs have a "xlate.ssc" suffix. For*

*more information on these ETDs, refer to the e*Xchange Partner Manager User's Guide.* ***The HIPAA ETD Library includes only Java ETDs.***

Installation commits the templates to the **default** schema on the Registry Host that you specified during the installation process.

Within the relevant template directory, there are two files for each transaction:

- .**xsc** is the Java Event Type Definition file.
- .**jar** is the associated Java jar file.

## 3.3.2. File Names

File names for the templates are designed to assist you in quickly locating the file you want. Each file name consists of the same set of elements in the same sequence.

Because Addenda have been created for each of the eleven 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."

### HIPAA File Names

The file names for 4010 X12 Java HIPAA transactions are constructed as follows:

- The name of the standard, followed by an underscore
  - X12_
- 004010X092_: 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.
- Year designation:
  - 99_ for 1999 files
  - 00_ for 2000 files and Addenda files
- hipaa277_ An indicator that this is a HIPAA ETD, 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.

***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 for the transaction name; for example, HealCareClaiPaym for Health Care Claim Payment
- .xsc (file extension)

Examples:

- The file name for a 277, Health Care Claim Status Notification, for HIPAA 1999 is **X12_004010X093_99_hipaa277_HealCareClaiStatNoti.xsc**
- The file name for a 270, Eligibility Coverage or Benefit Inquiry, for HIPAA 2000 is **X12_004010X092_00_hipaa270_EligCoveOrBeneInqu.xsc**
- The file name for a 270, Eligibility Coverage or Benefit Inquiry Addendum for HIPAA 2000 is **X12_004010X092A1_00_hipaa270_EligCoveOrBeneInqu.xsc**

## NCPDP-HIPAA File Names

The file names for NCPDP Java HIPAA transactions are constructed as follows:

- NCPDP_ (name of standard followed by underscore)
- T51_, Batch11, or Batch_1_0 (version type and number; for Telecom 5.1, version type for Telecom is followed by underscore)

The following characters are used for Telecom transactions only.

- Two-character transaction code followed by underscore; for example, E1_ or N3_ (indicates transaction type, such as Eligibility Verification or Information Reporting Rebill)
- REQ_ or RESP_ (indicates whether the message is a request or a response)
- For responses only, a single-character code indicating the type of response followed by an underscore; for example, 4_.
- Abbreviation for the transaction name; for example, BillRequ for Billing Request.
- .xsc (file extension)

For example:

- The Java file name for a Prior Authorization Inquiry Response: Transmission Accepted; Transaction Rejected is **NCPDP_T51_P3_RESP_4_PAInquRespTranAcceReje.xsc**.
- The Java file name for a Batch 1.1 transaction is **NCPDP_Batch11**.

## Addenda Files

Amendments to the 004010 Implementation Guides of May 2000 were approved by the X12N Health Care Work Group in October 2002 and published in February 2003. These Addenda are optional until October 15, 2003 but required after that date.

ESR (software patch) 53310 does the following:

- Removes HIPAA validations from the X12 4010 HIPAA Standard ETDs to pave the way for validation through a network appliance from Claredi Corporation.

- Adds the X12 4010 HIPAA Addenda ETDs to the HIPAA_2000 subdirectory. Addenda ETDs supplement the ETDs for which they are amendments, and function the same way.

Addenda ETD files are as follows:

```
X12_004010X061A1_00_hipaa820_PaymOrdeAdvi.jar
X12_004010X061A1_00_hipaa820_PaymOrdeAdvi.xsc
X12_004010X091A1_00_hipaa835_HealCareClaiPaym.jar
X12_004010X091A1_00_hipaa835_HealCareClaiPaym.xsc
X12_004010X092A1_00_hipaa270_EligCoveOrBeneInqu.jar
X12_004010X092A1_00_hipaa270_EligCoveOrBeneInqu.xsc
X12_004010X092A1_00_hipaa271_EligCoveOrBeneInfo.jar
X12_004010X092A1_00_hipaa271_EligCoveOrBeneInfo.xsc
X12_004010X093A1_00_hipaa276_HealCareClaiStatRequ.jar
X12_004010X093A1_00_hipaa276_HealCareClaiStatRequ.xsc
X12_004010X093A1_00_hipaa277_HealCareClaiStatNoti.jar
X12_004010X093A1_00_hipaa277_HealCareClaiStatNoti.xsc
X12_004010X094A1_00_hipaaA1_278_HealCare ServReviInfo.jar
X12_004010X094A1_00_hipaaA1_278_HealCareServReviInfo.xsc
X12_004010X094A1_00_hipaaA3_278_HealCareServReviInfo.jar
X12_004010X094A1_00_hipaaA3_278_HealCareServReviInfo.xsc
X12_004010X095A1_00_hipaa834_BeneEnroAndMain.jar
X12_004010X095A1_00_hipaa834_BeneEnroAndMain.xsc
X12_004010X096A1_00_hipaaQ3_837_HealCareClai.jar
X12_004010X096A1_00_hipaaQ3_837_HealCareClai.xsc
X12_004010X097A1_00_hipaaQ2_837_HealCareClai.jar
X12_004010X097A1_00_hipaaQ2_837_HealCareClai.xsc
X12_004010X098A1_00_hipaaQ1_837_HealCareClai.jar
X12_004010X098A1_00_hipaaQ1_837_HealCareClai.xsc
```

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**

Chapter 4

# Working With the HIPAA X12 ETDs

This chapter provides information on additional features built into the X12 ETDs, and includes instructions on working with and testing the ETDs. This chapter also provides information on using the custom Java methods provided within the ETDs, and other general information about using the X12 ETD Library.

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

## 4.1 HIPAA ETD Components Naming Conventions

Each HIPAA ETD contains envelope, transaction, segment loop, segment, and element names. In addition, there may be mask names and composite names. The components in an ETD correspond to the components in the X12 transaction type represented by the ETD. 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 ETD 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 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 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 ETDs, 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 ETDs 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 ETDs 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.

## 4.2 Customizing a Java ETD

Currently eGate Integrator does not support the editing of pre-built Java ETDs. However, the e*Gate Integrator offers a feature that allows you to convert existing Monk ETDs (**.ssc** files) to Java-enabled ETDs (**.xsc** files). This feature is the SSC Wizard.

**To create a customized Java ETD**

1. Create a corresponding Monk ETD, or use the Monk version of the Java ETD if available.
2. Customize the Monk ETD (**.ssc** file) using the e*Gate ETD Editor.
3. Convert the Monk ETD to a Java ETD using the e*Gate SSC Wizard.

When the conversion is done, you have three files:

- The original Monk ETD (**.ssc** file)
  Keep this file in case further customization is needed.
- The Java version of the ETD (**.xsc** file)
- The corresponding **.jar** file

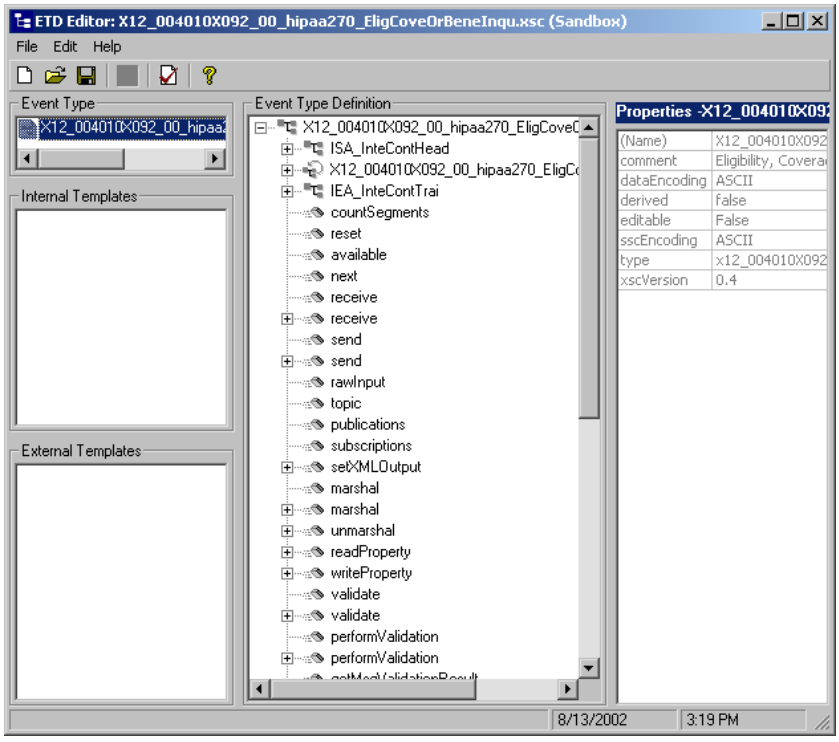
If you need to make further changes to the ETD, make the changes in the **.ssc** file and run the conversion again.

For specific instructions on using the e*Gate ETD Editor or the SSC Wizard, refer to the *e*Gate Integrator User's Guide*.

## 4.3 Viewing a HIPAA X12 ETD in the ETD Editor

An example of a HIPAA 270 transaction in the Java ETD Editor is shown in Figure 4.

**Figure 4** HIPAA 270 In the ETD Editor




The ETD shown in Figure 4 is **X12_004010X092_00_hipaa270_EligCoveOrBeneInqu.xsc**. The root node is **x12_004010X092_00_hipaa270_EligCoveOrBeneInquOuter**. For each X12 ETD, the root node name is the same as the file name, but without the extension and with **Outer** appended to the file name.

Some things to note about X12 Java ETDs:

- Bubble text labels are available for some of the items.
- In the .**xsc** file, the following naming conventions apply:
  - An element name begins with **E**
  - A segment loop name begins with **Loop**

## 4.4 Setting the Delimiters

The HIPAA X12 ETDs must include some way for delimiters to be defined so that they can be mapped successfully from one ETD to another.

In X12, delimiters are specified in the interchange header segment (ISA).

The delimiters are as follows:

- Data Element Separator (default is an asterisk)
- Subelement Separator/Component Element Separator (default is a colon)
- Segment Terminator (default is a tilde)

These delimiters can be set 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 Collaboration Editor by means of Java methods that are provided in the ETD files.

For specific information on the Java methods provided for the getting and setting of delimiters, refer to **"HIPAA ETD Library Java Methods" on page 31**.

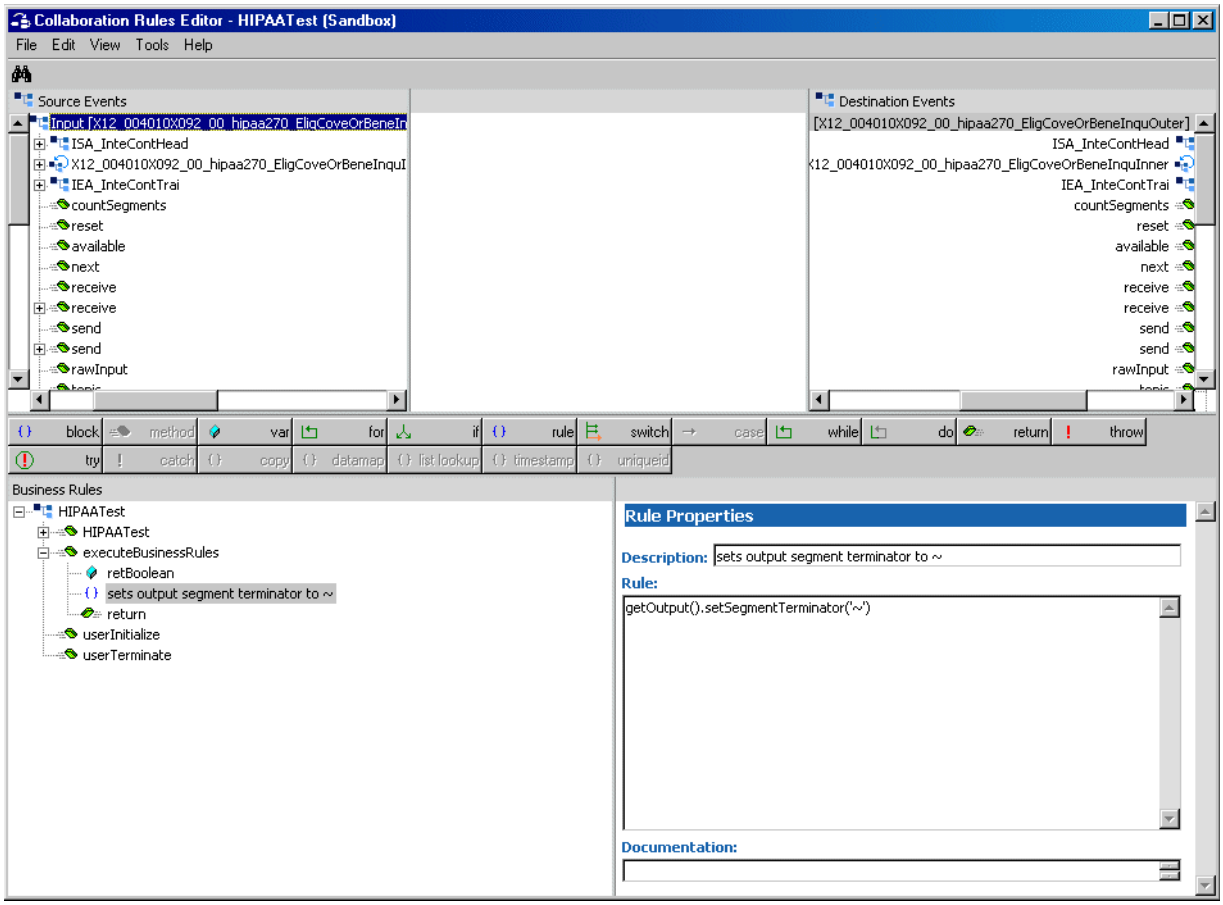
If the input data is already in HIPAA X12 format, you can use the "get" methods to get the delimiters from the input data. If the Collaboration is putting the data into HIPAA X12 format, you can use the "set" methods to set the delimiters in the output ETD.

### To set the delimiters in the Collaboration Rules Editor

1. Open the Collaboration in the Java Collaboration Rules Editor.
2. In the **Business Rules** section, add a rule.
3. Click on the method that you want to use.
4. Drag and drop the method to the **Rule Properties** section (an example is shown in Figure 5).

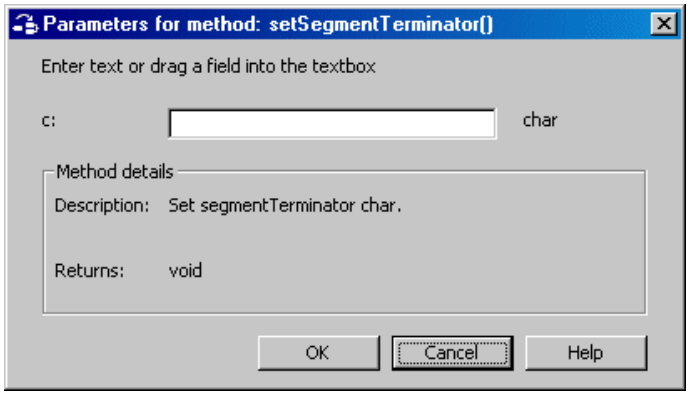
***Note:*** *When building Collaboration Rules scripts with Java ETDs, if there is data mapped to a field in a Java template and there are optional fields on the same level with no data mapped to them, the output includes delimiters for the optional fields.*

**Figure 5** Setting Delimiters in a Collaboration Rules Component



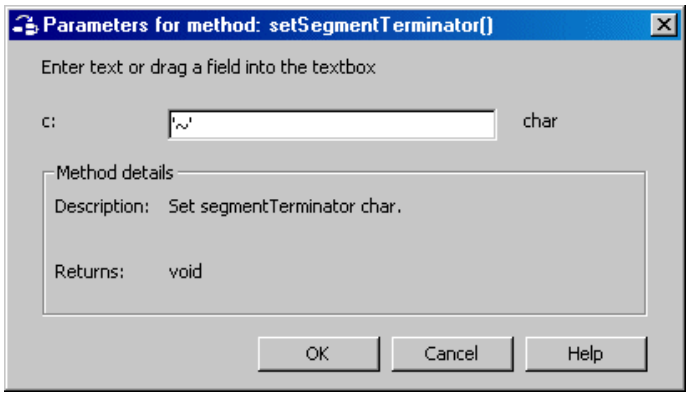

5 The **Parameters for method:** dialog box appears (see Figure 6).

**Figure 6** Parameters for Method Dialog Box




6 Set the delimiter value (an example is shown in Figure 7).

**Figure 7** Parameters for Method Dialog Box Showing Delimiter Value




7 Click **OK**.

8 Save the Collaboration Rules component.

***Note:*** *You **must** specify the delimiters. You can do this either by setting individual delimiters to specific values, or by using the setdefaultX12delimiters Java method to set the defaults.*

## 4.5 Running Validation in the Collaboration Rules Component

An additional tool you can use for validating your data is to run validation methods within the Collaboration.

The HIPAA X12 ETD Library includes several Java methods provided for this purpose. They are as follows:

- performValidation
- performValidation(boolean)
- getMsgValidationResult
- getICValidationResult
- getFGValidationResult
- getTSValidationResult

For more information on these Java methods, refer to **"HIPAA ETD Library Java Methods" on page 31**.

### 4.5.1. Java Collaboration Rules

If you install e*Xchange, a set of predefined Java validation Collaboration Rules is installed to validate transactions that use the ETDs from the HIPAA ETD Library. You

can use these Collaboration Rules to implement a comprehensive, tested HIPAA solution, or you can create your own Collaboration Rules to validate the ETD.

## HIPAA Collaboration Rules

The HIPAA Collaboration Rules provided with e*Xchange provide HIPAA validations in addition to those provided in the HIPAA ETDs. There is one Collaboration Rule for each transaction type, and each Collaboration Rule was designed to work with a specific HIPAA ETD. The output of the Collaboration Rules is defined by the ETD **X12ValidationResult**, and includes any errors that were found during the validation process.

The validation method nodes in an **.xsc** file are used to validate a HIPAA X12 message at run time. The methods return arrays containing descriptions about any invalid data elements, segments, segment loops, envelopes, and so forth.

The **performValidation** method performs specific validations on the Interchange Group, Functional Group, and Transaction Set envelopes as follows, and outputs any invalid information into an array (for more information about the format of the error output for envelope errors, see **"Error Message Formats" on page 59**). The following validations are performed against the message envelopes:

- Verifies that the control numbers in the ISA and IEA segments match.
- Verifies the number of transactions and verifies the number against the transaction count value provided in the GE01 segment of the Functional Group trailer (GE).
- Validates the transaction count by checking the number of transactions against the count provided. The transaction is invalid if the numbers do not match.
- Verifies that the number of segments listed matches the actual count.
- Checks for missing or invalid Transaction Set headers.

The results of the validation method can be retrieved by calling error retrieval methods (described in Chapter 5 of this guide). You can choose how to direct the output of the string; for example, to a log file.

***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.*

## Creating a Collaboration Rule to Validate the ETD

The elements that are part of an **.xsc** file can be dragged and dropped when two or more **.xsc** files are opened in the Collaboration Rules Editor (see the *e*Gate Integrator User's Guide* for more information). A field in the **Source** pane can be dragged to a field in the **Destination Events** pane. This action, when highlighted in the **Business Rules** pane, displays the rule in the **Rule Properties** pane.

# 4.6 Alternative Formats: ANSI and XML

By default, all the HIPAA X12 ETDs accept either standard ANSI X12 format or XML format as input; no changes to the existing Collaborations are needed. However, if you are using ETDs from previous releases, you must recompile your Collaborations after installing the HIPAA X12 ETD Library. The package names for the Java methods have changed for the ETDs.

By default, output is ANSI. However, there are two Java Methods available for setting the output to XML.

## 4.6.1. XML Format for HIPAA X12

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

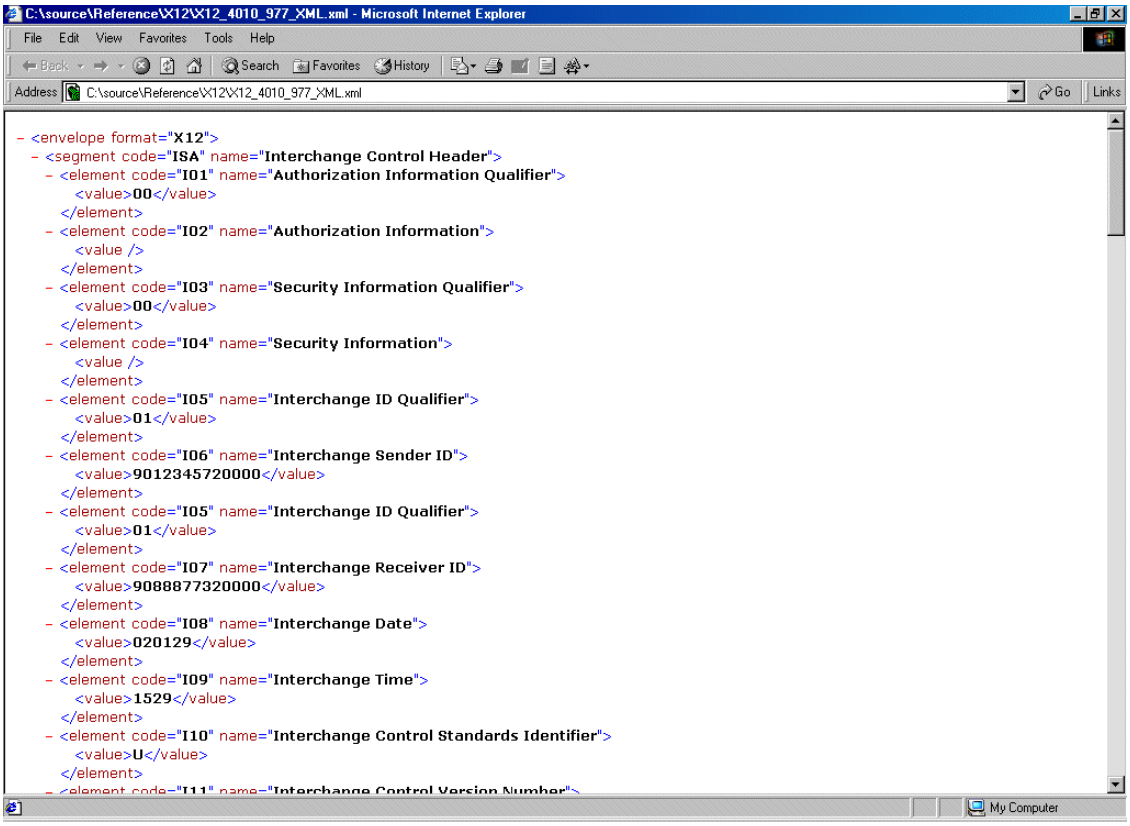
The XML X12 DTD is shown in Figure 8.

**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>
```

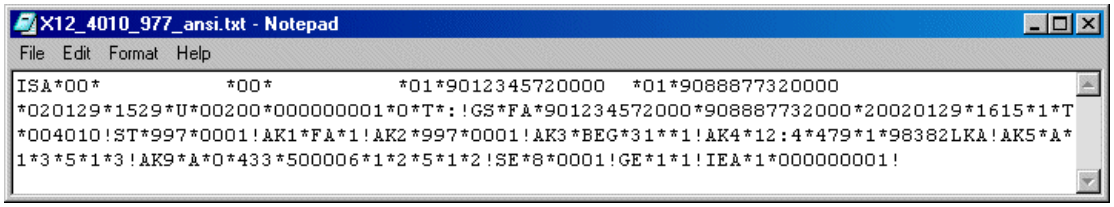
Figure 9 shows an X12 997 Functional Acknowledgment, in XML format.

**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




## 4.6.2. Setting the Collaboration to XML Output

By default, output from a Collaboration that uses standard Events from the HIPAA X12 ETD Library is in ANSI X12 format.

If you want to set the Collaboration to output XML format, use one of the following two new Java methods:

- **setXMLOutput** (*boolean isXML*) with the argument set to true if the outbound HIPAA X12 ETD is set to automatically publish.
- **marshal** (*boolean isXMLOutput*) with the argument set to true if the outbound HIPAA X12 ETD is set to manually publish.

Figure 11 shows a HIPAA X12 Collaboration. A rule is being added to the Collaboration to set the output to XML.

**Figure 11** Setting the Output to XML in the HIPAA X12 Collaboration

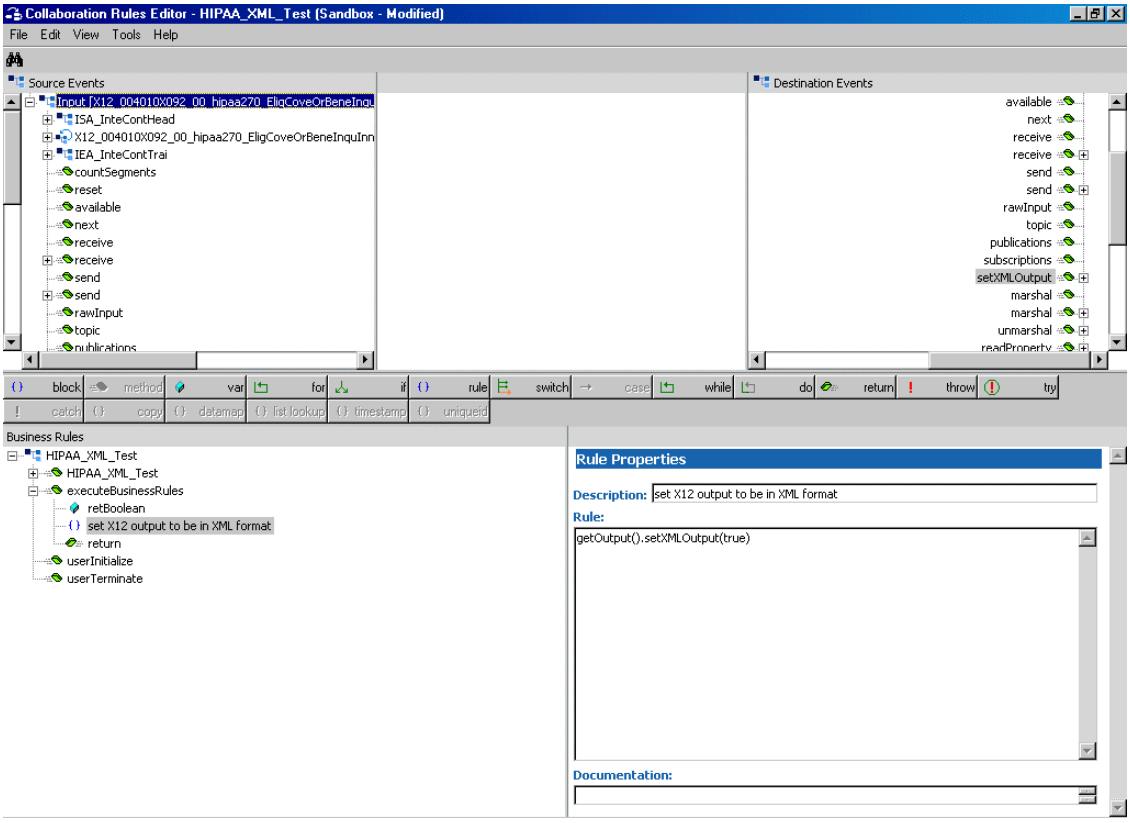
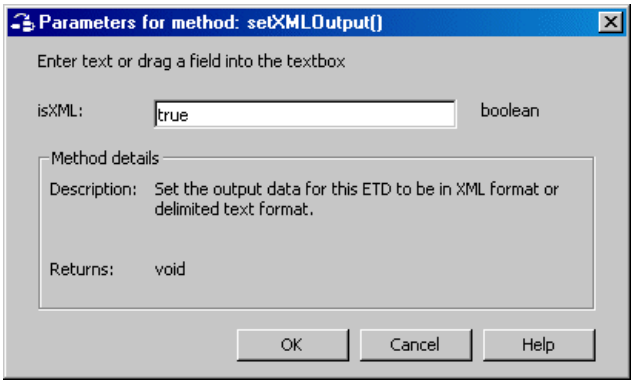



Figure 12 shows the parameter for **setXMLOutput** ( ) being set.

**Figure 12** Specifying the Parameter for **setXMLOutput** ( )



# 4.7 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

# HIPAA ETD Library Java Methods

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

## 5.1 Java Methods

### 5.1.1. Overview

The HIPAA ETD Library provides methods for a variety of purposes, including validation and error reporting, retrieving and setting delimiters, and defining HIPAA mandate national identifier processing. Error processing is divided into three types: marshalling errors, message errors, and envelope errors. Message validation is typically performed after unmarshalling, which means after the input data has been parsed and populated into the ETD structure. However, you can also validate the data from memory by calling **performValidation** with the parameter set to "false". After data is unmarshalled and validated, it is then "marshalled", meaning output data is generated from the validated ETD structure.

### 5.1.2. Available Java Methods

The HIPAA ETD Library provides methods that allow you to retrieve the standard X12 delimiters from the input ETD and set them appropriately for the output ETD; or to set the delimiters to the defaults. The methods are:

- **setRepetitionSeparator** on page 37

The HIPAA ETD Library also includes the following custom Java methods for testing the validation Collaborations and outputting error message information:

- **performValidation (no parameters)** on page 38
- **performValidation (boolean parameter)** on page 39
- **isUnmarshalComplete** on page 40
- **getUnmarshalErrors** on page 40
- **getMsgValidationResult** on page 41
- **getAllErrors** on page 42
- **getICValidationResult** on page 43
- **getFGValidationResult** on page 44
- **getTSValidationResult** on page 44
- **validate (no parameters)** on page 45
- **validate (boolean parameter)** on page 46

The library includes the following utility method to count the segments in a given transaction:

- **countSegments** on page 47

The HIPAA ETD library includes the following methods for setting the output of a Collaboration to XML:

- **setXMLOutput (boolean isXML)** on page 47
- **marshal (boolean isXMLOutput)** on page 48

Three methods are provided in the HIPAA ETD library specifically to assist in creating post-validation processing Collaboration Rules.

- **stripDataError** on page 49
- **addUserDataError** on page 51
- **isExternalCode** on page 52

Finally, the library includes the following methods for getting and setting the unique identifier flags for payors, providers, individuals, and employers:

- **getMandatePlanId** on page 53
- **setMandatePlanId** on page 54
- **getMandateProviderId** on page 55
- **setMandateProviderId** on page 55
- **getMandateIndividualId** on page 56
- **setMandateIndividualId** on page 57
- **getMandateEmployerId** on page 57
- **setMandateEmployerId** on page 58

## setDefaultX12Delimiters

### Description

Sets the default X12 delimiters.

### Syntax

```
public void setDefaultX12Delimiters()
```

### Parameters

None.

### 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();
......
......
input.setDefaultX12Delimiters();
```

## 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);
```

## performValidation (no parameters)

### Description

Performs validation on the ETD content after unmarshalling from the input data. Unlike the **validate** method, **performValidation** does not return a string of error messages. After calling **performValidation**, you must call **getAllErrors**, **getMsgValidationResult**, **getICValidationResult**, **getFGValidationResult**, or **getTSValidationResult** to obtain error information for the input data.

### Syntax

```
public void performValidation()
```

### Parameters

None.

### 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();
com.stc.hipaa.X12ValidationResult output=new
com.stc.hipaa.X12ValidationResult();
.....
.....
if (input().isUnmarshalComplete())
   input().performValidation();
   output().addDataErrors(input().getMsgValidationResult());
```

## performValidation (boolean parameter)

### Description

Validates the ETD content, either immediately after unmarshalling or in memory. Unlike the **validate** method, **performValidation** does not return a string of error messages. You must call **getAllErrors**, **getMsgValidationResult**, **getICValidationResult**, **getFCValidationResult**, or **getTSValidationResult** to obtain error information for the input data.

When used with the parameter set to true, this method works in the same way as **performValidation** (with no parameters). When set to false, this method validates the input in memory (that is, the input prior to unmarshalling). For more information about validating in memory or validating after unmarshalling, see **"validate (boolean parameter)" on page 46**.

### Syntax

```
public void performValidation(boolean original)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| original | boolean | If true, validates the ETD content right after unmarshalling. If false, validates the ETD in memory. |

### 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();
com.stc.hipaa.X12ValidationResult output=new
com.stc.hipaa.X12ValidationResult();
.....
.....
if (input().isUnmarshalComplete())
   input().performValidation(true);
   output().addDataErrors(input().getMsgValidationResult());
```

## isUnmarshalComplete

### Description

Checks if unmarshalling was completed successfully. If unmarshalling was completed successfully, processing continues. If it was not completed successfully, call **getUnmarshalErrors** to retrieve a list of the errors that occurred.

### Syntax

```
public boolean isUnmarshalComplete()
```

### Parameters

None.

### Constants

None.

### Returns

**boolean**
Returns Boolean true if unmarshalling was completed successfully; returns Boolean false otherwise.

### 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());
```

## 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 59**.

### 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 59***.*

### 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 59***.*

### 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());
```

## validate (no parameters)

### Description

Validates the ETD content in memory, and outputs a concatenation of strings listing all unmarshalling, envelope, and message errors that occurred.

For example, if one of the nodes populated in the ETD has an inappropriate value, this method outputs a string providing this information. If there are no problems with the ETD content, the output is a null string.

If you want to list validation errors by type of error instead of listing all errors together, use **performValidation**, along with the message output methods (**getMsgValidation**, **getTCSValidationResult**, and so on), instead of **validate**.

### Syntax

```
public java.lang.String validate()
```

### Parameters

None.

### Constants

None.

### Returns

**String**
A description of the errors in the data. If there are no errors, the string is null.

### 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();
......
......
string msg=input.validate();
```

## validate (boolean parameter)

### Description

Validates the ETD content, either immediately after unmarshalling or in memory, depending on the value of the Boolean parameter. This method outputs a concatenation of strings containing all error types, including marshalling, message, and envelope errors. If you want to list validation errors by type of error instead, use **performValidation,** along with the message output methods (**getMsgValidation**, **getTCSValidationResult**, and so on), instead of **validate**.

When used with the parameter set to false, this method works in the same way as validate (with no parameters); that is, it validates the ETD content in memory. When the parameter is set to true, this method validates the input data file after unmarshalling to the ETD. When the parameter is set to true, this method is able to provides more extensive validation.

```
1*BHT*2**8*4**373*4*980905*BHT_4 at 2 [980915]: Data has too few
characters of 6 because >= 8.
```

### Syntax

```
public java.lang.String validate(boolean original)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| original | boolean | If true, validates the ETD content right after unmarshalling. If false, validates the ETD in memory. |

### Constants

None.

### Returns

**java.lang.String**
A description of the errors in the data. If there are no errors, the string is null.

### 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();
......
......
string msg=input.validate(true);
```

## countSegments

### Description

Returns a count of segments in the given level.

### Syntax

```
public int countSegments()
```

### Parameters

None.

### countSegments Constant

None.

### Returns

**int**

An integer representing the number of segments counted in the given level.

### 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();
......
......
int segments=input.countSegments();
```

## setXMLOutput (boolean isXML)

### Description

When used with the parameter set to true, this method causes the HIPAA ETD involved to output XML.

When used with the parameter set to false, this method causes the HIPAA ETD to output ANSI (which is the default output if this method is not used at all).

Use this method when the HIPAA ETD is set to automatic output (the default). If the Collaboration is set to manual output, use **marshal** (*boolean*) to achieve the same result.

### Syntax

```
public void setXMLOutput(boolean isXML)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| isXML | boolean | If true, the HIPAA X12 is output in XML format. If false, output is standard ANSI X12. |

### 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();
......
......
input.setXMLOutput(true);
```

## marshal (boolean isXMLOutput)

### Description

When used with the parameter set to true, this method generates the output byte array in XML format.

When used with the parameter set to false, this method generates the output byte array in ANSI format.

Use this method when the ETD is set to manual output. If the ETD is set to automatic output (the default), use **setXMLOutput** (*boolean parameter*) to achieve the same result.

### Syntax

```
public byte[] marshal(boolean isXMLOutput)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| isXMLOutput | boolean | If true, the HIPAA X12 is output in XML format. If false, output is standard ANSI X12. |

### Constants

None.

### Returns

**byte []**
The output in byte array format.

### 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();
......
......
byte[] output=input.marshal(true);
```

## stripDataError

### Description

Removes error data from the transaction. Use this method to make the validations against certain elements in HIPAA transactions less restrictive.

### Syntax

```
public void stripDataError(int ErrorCode, short level,
java.lang.String loopIDCode, java.lang.String segmIDCode, int
segmPosiInTransSet, short segmSyntErroCode, short elemPosiInSegm,
short compDataElemPosiInComp, java.lang.String dataElemRefNumb, short
dataElemSyntErroCode, java.lang.String CopyOfBadDataElem)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| ErrorCode | int | The error code number for the error message to be stripped. ***Note:*** *In previous versions, HIPAA error code numbers began at 5000; for the current version, they begin at 15000.* |
| level | short | The Claredi level of the error message. |
| loopIDCode | java.lang.String | The loop identifier of the loop in which the error data is located. |
| segmIDCode | java.lang.String | The segment identifier of the segment in which the error data is located. |
| segmPosiInTransSet | int | The position of the segment in the transaction set. |

| Name | Type | Description |
|---|---|---|
| segmSyntErroCode | short | The segment syntax error as it appears in the AK304 segment of the 997 Functional Acknowledgment. |
| elemPosiInSegm | short | The position of the element within the segment. |
| compDataElemPosiInComp | short | The position of the element in the composite. |
| dataElemRefeNumb | java.lang.String | The reference number of the data element. |
| dataElemSyntErroCode | short | The element error code as it appears in the AK403 segment of the 997 Functional Acknowledgment. |
| CopyOfBadDataElem | java.lang.String | A copy of the bad data value. |

### Parameter Notes

When defining parameters, follow these guidelines.

- For parameters of the data type **short**, you must use a cast for the parameter value. For example, if the parameter value is "3", the parameter must be defined as "(short)3" (without the double quotes).
- For parameters of the data type **java.lang.String**, the value of the parameter must be placed in double quotes. For example, "Element value is not valid."
- To specify that a parameter not be used in a specific call to a method:
  - Enter **-1** for parameters of the type **int**.
  - Enter **(short)-1** for parameters of the type **short**.
  - Enter **null** (the actual text "null" with no quotes) for parameters of the type **java.lang.String**.

  For example, if you do not know the Claredi level of the message you want to strip, enter "(short)-1" (no quotes) for the **level** parameter for **stripDataError**. The method will then ignore the Claredi level when searching for errors to remove.

### Returns

void (none).

### Throws

None.

### Examples

```
getoutput().stripDataError(15004,(short)-1,"2000B","SBR",14,
(short)-1,(short)4,(short)-1,null,(short)-1,null);
getoutput().stripDataError(15008,(short)-1,"1000A","PER",-1,
(short)-1,(short)-1,(short)-1,null,(short)-1,null);
```

## addUserDataError

### Description

Adds custom error data to the transaction. Use this method to create custom error messages for the validations you define for reprocessing HIPAA transactions.

***Note:*** *When defining the error messages for the validation logic you create, you must follow the format of the e*Xchange HIPAA error message structure. For more information about these messages, see "Validation Error Reporting" and "Appendix C: Error Codes" in the HIPAA Implementation Guide.*

### Syntax

```
public void addUserDataError(int errorCode, java.lang.String
errorDesc, short level, java.lang.String loopIDCode, java.lang.String
segmIDCode, int segmPosiInTransSet, short segmSyntErroCode, short
elemPosiInSegm, short compDataElemPosiInComp, java.lang.String
dataElemRefNumb, short dataElemSyntErroCode, java.lang.String
CopyOfBadDataElem)
```

### Parameters

***Important:*** *Review the* **Parameter Notes** *on page 50* *before using the following parameters.*

| Name | Type | Description |
|---|---|---|
| errorCode | int | The error code number for the error message to be added.<br>***Note:*** *The error code numbers must be between 20000 and 29999.* |
| errorDesc | java.lang.String | A short description of the error that occurred. This string cannot contain pipes ("\|") or asterisks ("*"). |
| level | short | The Claredi level of the error message. |
| loopIDCode | java.lang.String | The loop identifier of the loop in which the error data is located. |
| segmIDCode | java.lang.String | The segment identifier of the segment in which the error data is located. |
| segmPosiInTransSet | int | The position of the segment in the transaction set. |

### Returns

void (none).

### Throws

None.

### Examples

```
getoutput().addUserDataError((short)3,29001,"This is not a valid
value.","2010AB","NM1",27,(short)2,(short)1,(short)7,"576",(short)6,
"Bad Data");
getoutput().addUserDataError((short)3,29002,"Value not found in
specified codeset.","2010AB","NM1",27,(short)8,(short)1,(short)7,
"576",(short)6,"Bad Data")
```

## isExternalCode

### Description

Specifies an external code set to use to validate the specified element. Use this method to change the code set used for a specific element. Be sure to strip any HIPAA code set errors written against the specified element before re-validating the element against a new code set.

### Syntax

The **isExternalCode** method can use four different sets of parameters. If you do not specify a date in the parameter list, the method uses the most recent version of the code set.

```
public final static boolean isExternalCode(String value, String
sources)

public final static boolean isExternalCode(String value, String
sources, String dt)

public final static boolean isExternalCode(String value, String
sources, String dt, String qual)

public final static boolean isExternalCode(String value, String
sources, java.util.Date dt)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| value | java.lang.String | The value or element to validate using the specified code set. |
| sources | java.lang.String | The name of the code set to use to validate the specified element. This name must match a code set name defined in the External Codeset Configuration function. |
| dt | java.lang.String | The date in string format. If no **qual** parameter is specified, the format must be "YYYYMMDD". If a **qual** parameter is specified, the format is specified by the qualifier (**qual**). |

| Name | Type | Description |
|---|---|---|
| segmSyntErroCode | short | The segment syntax error as it appears in the AK304 segment of the 997 Functional Acknowledgment. |
| elemPosiInSegm | short | The position of the element within the segment. |
| compDataElemPosiInComp | short | The position of the element in the composite. |
| dataElemRefeNumb | java.lang.String | The reference number of the data element. |
| dataElemSyntErroCode | short | The element error code as it appears in the AK403 segment of the 997 Functional Acknowledgment. |
| CopyOfBadDataElem | java.lang.String | A copy of the bad data value. |
| dt | java.util.Date | The date in java.lang.Date format. This can be used in place of a date in string format, but cannot be used if the **qual** parameter is specified. |
| qual | java.lang.String | A date format qualifier as defined in SEF. |

### Returns

**boolean**
Returns Boolean true if the specified element passed the validation against the new code set; returns Boolean false otherwise.

### Throws

None.

### Examples

The following sample defines a new external code set, **51**, to use to revalidate the value of the PostCode element.

```
......
if(!Util.isExternalCode(getinput().getX12.004010X096_00_hipaaQ3_837_H
ealCareClaiInner(0).getX12_004010X096_00_hipaaQ3_837_HealCareClai(0).
getLoop2000A_7_2000(0).getLoop2010AB_16_2010().getN4_msk1_18_GeogLoca
().getE116_3_PostCode(),"51"))
.....
```

## getMandatePlanId

### Description

Gets the mandate flag for HIPAA plan identifiers.

### Syntax

```
public boolean getMandatePlanId()
```

### Parameters

None.

### Constants

None.

### Returns

**boolean**
Returns true if the flag is set to validate per the HIPAA mandate; returns false if the flag is not set to validate per the HIPAA mandate.

### Throws

None.

### Examples

```
com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_HealCareClaiStatNoti
 Outer input=new com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_
 HealCareClaiStatNotiOuter();
......
......
boolean planID=input.getMandatePlanId();
```

## setMandatePlanId

### Description

Sets the mandate flag for HIPAA plan identifiers.

### Syntax

```
public void setMandatePlanId(boolean flag)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| flag | boolean | An indicator that specifies whether to validate against the HIPAA identifier mandate. Set the flag to true to validate plan identifiers; set the flag to false to allow non-mandated identifiers. |

### Constants

None.

### Returns

void (none).

### Throws

None.

### Examples

```
com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_HealCareClaiStatNoti
 Outer input=new com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_
 HealCareClaiStatNotiOuter();
......
......
boolean flag=true;
input.setMandatePlanId(flag);
```

## getMandateProviderId

### Description

Gets the mandate flag for HIPAA provider identifiers.

### Syntax

```
public boolean getMandateProviderId()
```

### Parameters

None.

### Constants

None.

### Returns

**boolean**
Returns true if the flag is set to validate per the HIPAA mandate; returns false if the flag is not set to validate per the HIPAA mandate.

### Throws

None

### Examples

```
com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_HealCareClaiStatNoti
 Outer input=new com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_
 HealCareClaiStatNotiOuter();
......
......
boolean providerID=input.getMandateProviderId();
```

## setMandateProviderId

### Description

Sets the mandate flag for HIPAA provider identifiers.

### Syntax

```
public void setMandateProviderId(boolean flag)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| flag | boolean | An indicator that specifies whether to validate against the HIPAA identifier mandate. Set the flag to true to validate provider identifiers; set the flag to false to allow non-mandated identifiers. |

### Constants

None.

### Returns

void (none).

### Throws

None.

### Examples

```
com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_HealCareClaiStatNoti
 Outer input=new com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_
 HealCareClaiStatNotiOuter();
......
......
boolean flag=true;
input.setMandateProviderId(flag);
```

## getMandateIndividualId

### Description

Gets the mandate flag for HIPAA patient identifiers.

### Syntax

```
public boolean getMandateIndividualId()
```

### Parameters

None.

### Constants

None.

### Returns

**boolean**

Returns true if the flag is set to validate per the HIPAA mandate; returns false if the flag is not set to validate per the HIPAA mandate.

### Throws

None.

### Examples

```
com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_HealCareClaiStatNoti
 Outer input=new com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_
 HealCareClaiStatNotiOuter();
......
......
boolean indivID=input.getMandateIndividualId();
```

## setMandateIndividualId

### Description

Sets the mandate flag for HIPAA patient identifiers.

### Syntax

```
public void setMandateIndividualId(boolean flag)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| flag | boolean | An indicator that specifies whether to validate against the HIPAA identifier mandate. Set the flag to true to validate patient identifiers; set the flag to false to allow non-mandated identifiers. |

### Constants

None.

### Returns

void (none).

### Throws

None.

### Examples

```
com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_HealCareClaiStatNoti
 Outer input=new com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_
 HealCareClaiStatNotiOuter();
......
......
boolean flag=true;
input.setMandateIndividualId(flag);
```

## getMandateEmployerId

### Description

Gets the mandate flag for HIPAA employer identifiers.

### Syntax

```
public boolean getMandateEmployerId()
```

### Parameters

None.

### Constants

None.

### Returns

**boolean**

Returns true if the flag is set to validate per the HIPAA mandate; returns false if the flag is not set to validate per the HIPAA mandate.

### Throws

None.

### Examples

```
com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_HealCareClaiStatNoti
 Outer input=new com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_
 HealCareClaiStatNotiOuter();
......
......
boolean employerID=input.getMandateEmployerId();
```

## setMandateEmployerId

### Description

Sets the mandate flag for HIPAA employer identifiers.

### Syntax

```
public void setMandateEmployerId(boolean flag)
```

### Parameters

| Name | Type | Description |
|---|---|---|
| flag | boolean | An indicator that specifies whether to validate against the HIPAA identifier mandate. Set the flag to true to validate employer identifiers; set the flag to false to allow non-mandated identifiers. |

### Constants

None.

### Returns

void (none).

### Throws

None.

### Examples

```
com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_HealCareClaiStatNoti
 Outer input=new com.stc.hipaa277_A120.X12_004010X093_00_hipaa277_
 HealCareClaiStatNotiOuter();
......
......
boolean flag=true;
input.setMandateEmployerId(flag);
```

## 5.2 Error Message Formats

The error messages you can retrieve using the HIPAA ETD Library Java methods vary in format depending on the type of messages you decide to retrieve.

### getAllErrors, getUnmarshalErrors, and getMsgValidationResult

These methods return messages in the format described in Chapter 4 of the HIPAA Implementation Guide under “Validation Error Reporting”. Additional error message information can be found in Appendix C, “Error Codes”, of the HIPAA Implementation Guide.

### getICValidationResult

This method retrieves error information for the interchange (IC) envelope validation. The resulting error messages contain five fields delimited by asterisks (*), and the information contained in the messages corresponds to certain fields in the X12 Interchange Acknowledgment response. Table 5 lists each field in the e*Xchange error message, providing a field description and the corresponding location in the Interchange Acknowledgment response for each. A sample error message appears below.

```
000001040*020428*0130*021*Invalid Number of Included Groups Value
```

**Table 4** Interchange Validation Error Message Format

| Error Message Field Position | Description | Interchange Acknowledgment Field |
|---|---|---|
| 1 | Interchange Control Number | TA101 |
| 2 | Interchange Date | TA102 |
| 3 | Interchange Time | TA103 |
| 4 | Interchange Note Code | TA105 |
| 5 | Interchange Note Description | NA |

e*Xchange reports the following interchange error notes. These note codes and descriptions appear in fields 4 and 5 of the e*Xchange error message.

- Note Code 001: The Interchange Control Number in the Header and Trailer Do Not Match. The Value From the Header is Used in the Acknowledgment.
- Note Code 021: Invalid Number of Included Groups Value

## getFGValidationResult

This method retrieves error information for the functional group (FG) envelope validation. The resulting error messages contain five fields delimited by asterisks (*), and the information contained in the messages corresponds to certain fields in the X12 997 Functional Acknowledgment response. Table 5 lists each field in the e*Xchange error message, providing a field description and the corresponding location in the 997 response for each. A sample message appears below.

```
HC*2*2*4*Group Control Number in the Functional Group Header and
Trailer Do Not Agree
```

**Table 5** Functional Group Validation Error Message Format

| Error Message Field Position | Description | 997 Response Field |
|---|---|---|
| 1 | Functional Identifier Code | AK101 |
| 2 | Group Control Number | AK102 |
| 3 | Number of Transaction Sets Included | AK902 |
| 4 | Functional Group Syntax Error Code | AK905 - AK909 |
| 5 | Functional Group Error Description | NA |

e*Xchange reports the following functional group errors. These error codes and descriptions appear in fields 4 and 5 of the e*Xchange error message.

- Syntax Error Code 4: Group Control Number in the Functional Group Header and Trailer Do Not Agree
- Syntax Error Code 5: Number of Included Transaction Sets Does Not Match Actual Count

## getTSValidationResult

This method retrieves error information for the transaction/message (TS) envelope validation. The resulting error messages contain four fields delimited by asterisks (*), and the information contained in the messages corresponds to certain fields in the X12 997 Functional Acknowledgment response. Table 6 lists each field, providing a field description and the corresponding location in the 997 response for each. A sample message appears below.

```
837*000001040*6*Missing or Invalid Transaction Set Identifier
```

**Table 6** Transaction/Message Validation Error Message Format

| Error Message Field Position | Description | 997 Response Field |
|---|---|---|
| 1 | Transaction Set Identifier Code | AK201 |
| 2 | Transaction Set Control Number | AK202 |
| 3 | Transaction Set Syntax Error Code | AK502 - AK506 |
| 4 | Transaction Set Error Description | NA |

e*Xchange reports the following transaction set errors. These error codes and descriptions appear in fields 3 and 4 of the e*Xchange error message.

- Syntax Error Code 3: Transaction Set Control Number in Header and Trailer Do Not Match
- Syntax Error Code 4: Number of Included Segments Does Not Match Actual Count
- Syntax Error Code 6: Missing or Invalid Transaction Set Identifier

Appendix A

# ASC X12 Overview

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

- An overview of ASC 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 e*Gate Integrator 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 8**.

## 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 e*Gate, a message structure is called an Event Type Definition (ETD). Each message structure (ETD) 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 X12 templates for a specific version includes transaction set structures for each of the transactions available in that version.

The X12 ETD Library provides e*Gate Event 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 X12 transaction. The HIPAA ETD Library provides a message structure for each X12 HIPAA 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 65**.

## 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 HIPAA ETD Library are the same as those recommended by the industry-specific implementation guides. These delimiters are shown in Table 7.

**Table 7** Default Delimiters in X12 ETD 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

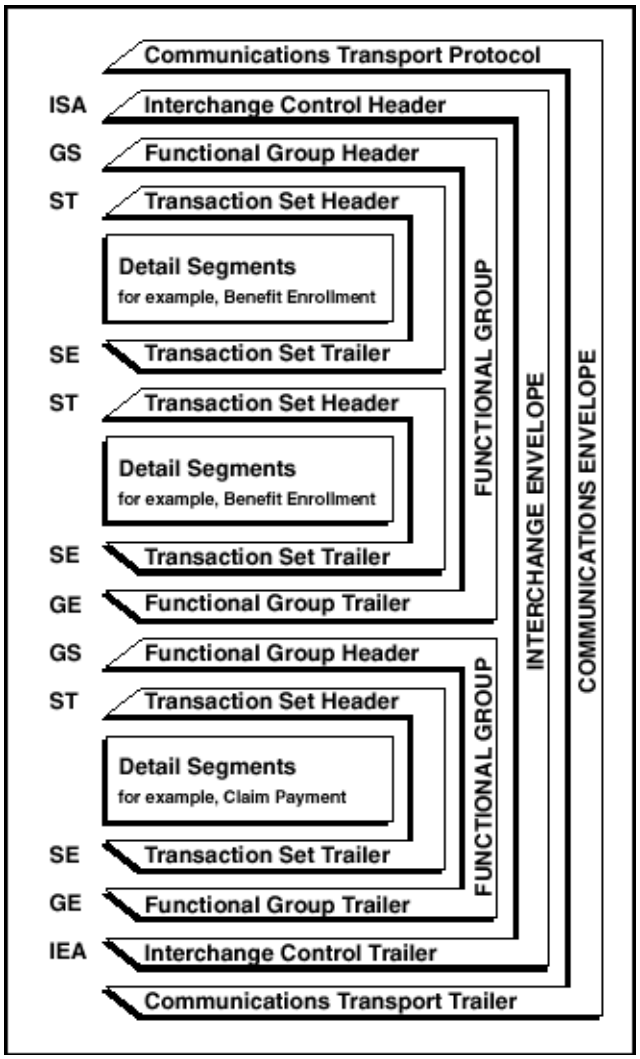
The rules applying to the structure of an X12 envelope are very strict to ensure the integrity of the data and the efficiency of the information exchange.

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 13. Each of these levels is explained in more detail in the following sections.

**Figure 13** X12 Envelope Schematic




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

Figure 14 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.

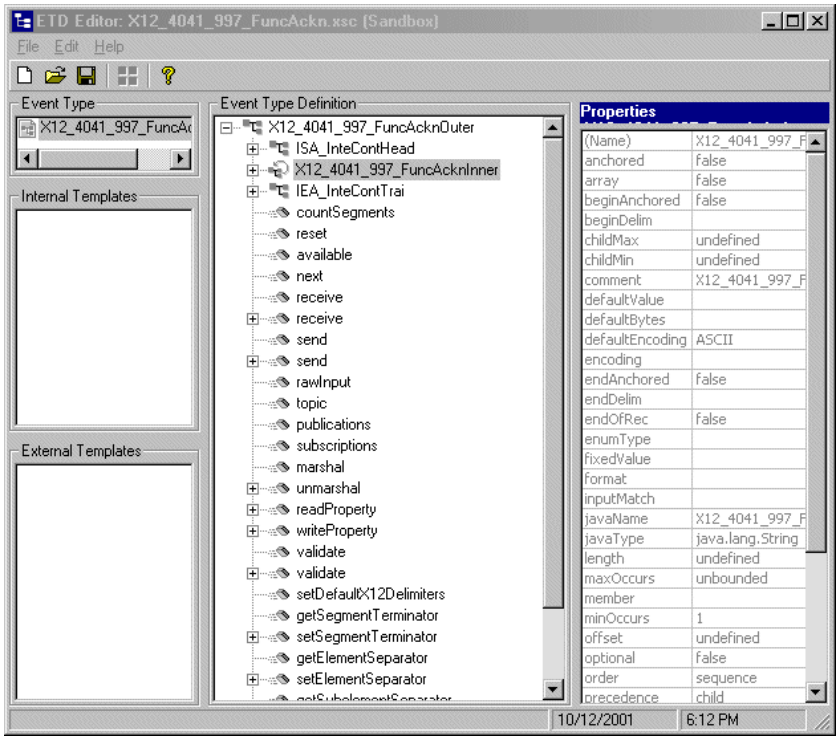
**Figure 14** X12 997 Segment Table

**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 15 shows the same transaction as viewed in the Java ETD Editor.

**Figure 15** X12 997 Viewed in Java ETD 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 16** Example of a Transaction Set Header (ST)

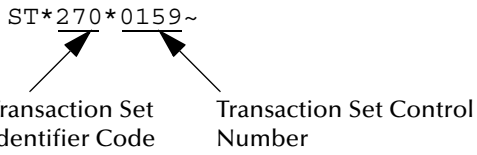
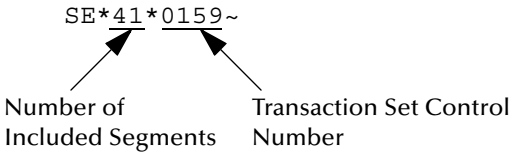



**Figure 17** Example of a Transaction Set Trailer (SE)




## 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 18** Example of a Functional Group Header (GS)

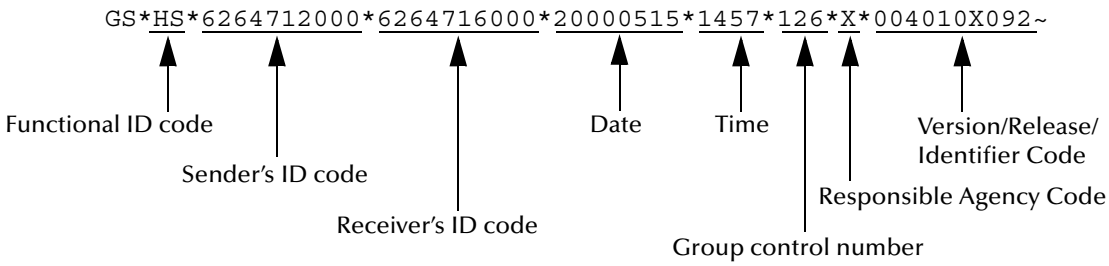
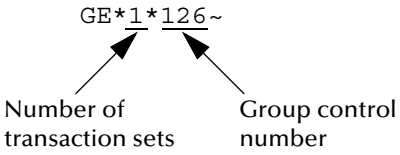



**Figure 19** Example of a Functional Group Trailer (GE)




## 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:

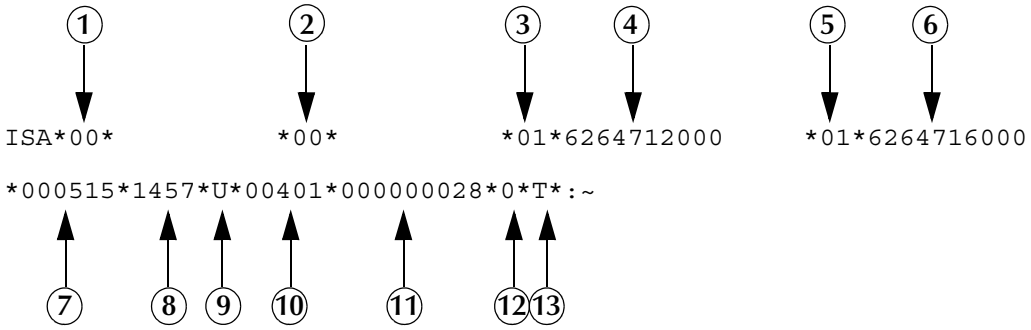
- Data element separators and data segment terminator
- Identification of sender and receiver
- Control information (used to verify that the message was correctly received)
- Authorization and security information, if applicable

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

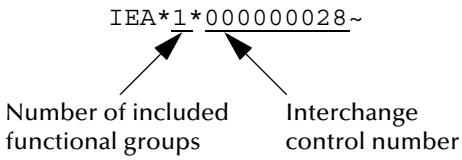
**Figure 20** Example of an Interchange Header (ISA)




Interchange Header Segments from Figure 20:

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 21** 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 8. The table describes each term, and lists its language analogy along with its associated e*Gate Collaboration scripts.

**Table 8** Key Parts of EDI Processing

| Term | Description | Language Analogy | e*Gate Collaboration Scripts |
|---|---|---|---|
| structures | format, segments, loops | syntax | ETD files or structures |
| validations | data contents “edit” rules | semantics | validation scripts |
| translations (also called mapping) | reformatting or conversion | translation | translation scripts |
| enveloping | header and trailer segments | envelopes | part of translation |
| acks | acknowledgments | return receipt | e*Way scripts |

e*Gate uses the structures, validations, translations, enveloping, and acknowledgments listed below to support HIPAA.

### A.5.1. Structures

The Event Type Definition library for HIPAA includes pre-built ETDs for all supported HIPAA versions.

### A.5.2. Validations, Translations, Enveloping, Acknowledgments

The e*Gate Integrator does not include any pre-built validations, transformations, or acknowledgments. These scripts can be built in the Java version of the Collaboration Rules Editor graphical user interface (GUI). These GUIs provide a user-friendly drag-and-drop front end for creating Java scripts.

For HIPAA, the e*Gate Integrator provides translations in Monk that add the enveloping information to the HIPAA message.

Installation of the e*Xchange Partner Manager includes a set of custom Java validations for HIPAA transactions, and also provides acknowledgment receipts, such as an X12 997 Functional Acknowledgment.

***Note:*** *In e*Gate, translations are called Collaborations.*

### A.5.3. 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 X12, visit the following Web sites:

- For X12 standard:

  **http://www.disa.org**

- For Implementation Guides: Washington Publishing Company at

  **http://www.wpc-edi.com**

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

# Index

## J



## L



## M



## N



## O



## P



## R



## S



## T

## V



## W



## X