SeeBeyond™ eBusiness Integration Suite

HIPAA ETD Library User's Guide

Release 4.5.2

SEEBEYOND™

Contents

Chapter 1

Introduction

This chapter introduces you to the HIPAA ETD Library User's Guide.

HIPAA is a standard that was developed specifically for the healthcare industry. For transactions related to health care, HIPAA uses a customization of X12. For pharmaceutical transactions, the HIPAA standard uses NCPDP (National Council for Prescription Drug Programs) transactions.

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

A general overview of X12 is also included, at the back of the book.

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 SeeBeyond eBusiness Integration Suite, and contains message definitions for HIPAA messages. This document gives a brief overview of HIPAA and the HIPAA message structures provided with e*Gate, 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 SeeBeyond™ eBusiness Integration Suite, to be thoroughly familiar with Windows 2000 and Windows NT operations and administration, and also with Microsoft Windows graphical user interfaces.

1.3 Compatible Systems

The e*Gate HIPAA ETD Library supports the following platforms:

- Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Windows NT 4.0 SP6a
- Solaris 2.6, 7, and 8
- AIX 4.3.3
- HP-UX 11.0 and 11i

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

1.4 Supporting Documents

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

- *X12 ETD Library User’s Guide*
- *NCPDP ETD Library User’s Guide*
- *e*Gate Integrator Installation Guide*

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 comprise 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 makes specifications affecting standards of treatment and privacy rights. It provides a number of standardized transactions that can be used for such things as a healthcare eligibility inquiry or a healthcare claim. HIPAA legislates that all of the healthcare industry will be on the same implementation timetable. All institutions doing electronic healthcare insurance transactions must implement these standardized transactions by the end of the year 2002.

More transactions will be added for later implementation.

HIPAA legislation mandates, among other items:

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

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

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

- Health insurance providers
- Provider management organizations

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

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

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

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

These transactions are based on the October 1997 X12 standard; that is, Version 4, Release 1, Sub-release 0 (004010) (version 4010).

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 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 2 NCPDP 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

Table 2 NCPDP Transaction Codes (Continued)

Code	Transaction Name
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

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. The specifics might vary between sites. The trading partner agreement is a useful repository for this type of site-specific information.

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

- The HIPAA standard

  HIPAA publishes a standard structure for each HIPAA 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.

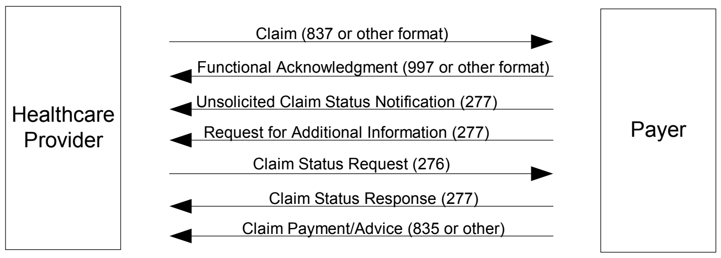
- Trading Partner Agreements

  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.

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


Healthcare Provider
Payer
Claim (837 or other format)
Functional Acknowledgment (997 or other format)
Unsolicited Claim Status Notification (277)
Request for Additional Information (277)
Claim Status Request (276)
Claim Status Response (277)
Claim Payment/Advice (835 or other)

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 38**.

Specifically, it uses the following elements:

- Segments

- Data elements
- Looping structures

2.1.6 Acknowledgment

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

The 997 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.hcfa.gov/HIPAA/HIPAAHM.HTM**
- **http://www.hipaa-dsmo.org**
- **http://www.wedi.org/**
- **http://www.ehnac.org/**
- **http://aspe.os.dhhs.gov/admnsimp/**

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

Chapter 3

HIPAA Template Installation

This chapter provides information on the installation procedure for the SeeBeyond 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
- NCPDP Telecom 5.0.1/Batch 1.0 and 1.1 transactions

Some additional points to note about the HIPAA transactions:

- The ETDs only accept messages with all the envelope segment information.
- Messages can be batched; however, all the messages in one functional group must be of the same message type.

3.1 HIPAA Libraries

When the HIPAA ETD Library is installed, there is a separate subdirectory for each set of transactions (three in total). Within each subdirectory all the files for that version are stored.

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 in on 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 you come to the **Please choose the product to install** dialog box.
4. Select **e*Gate Integrator**, and then click **Next**.
5. Follow the on-screen instructions until you come to the second **Please choose the product to install** dialog box.
6. Select **Add-ons**, and then click **Next**.
7. Follow the on-screen instructions until you come to the **Select Components** dialog box.
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 4.5.2**.
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 e*Gate installation, 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 for 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.

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.

3.3.1 HIPAA Folder Structure Created by Installation

By default, installation places the HIPAA 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
NCPDP	\eGate\Server\Registry\Repository\default\ETD\templates\NCPDP

Note: *Installation of e*Xchange includes a set of Monk HIPAA ETDs. 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.

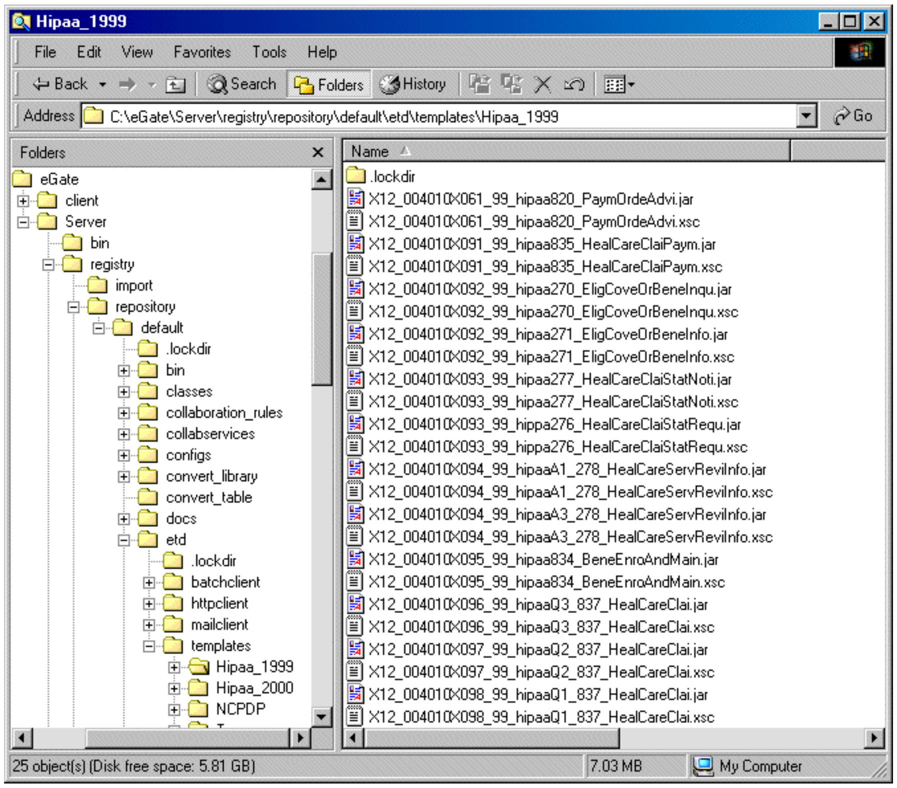
Within the relevant template directory, there are two files for each transaction:

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

Figure 2 shows an example of the directory structure for the HIPAA templates.

UNIX installation places the files in a similar folder structure.

Figure 2 HIPAA Version Folders


Hipaa_1999
File Edit View Favorites Tools Help
Back Search Folders History
Address
Go
Folders
Name

C:\eGate\Server\registry\repository\default\etd\templates\Hipaa_1999

```
eGate
  client
  Server
    bin
    registry
      import
      repository
        default
          .lockdir
          bin
          classes
          collaboration_rules
          collabservices
          configs
          convert_library
          convert_table
          docs
          etd
            .lockdir
            batchclient
            httpclient
            mailclient
            templates
              Hipaa_1999
              Hipaa_2000
              NCPDP
```

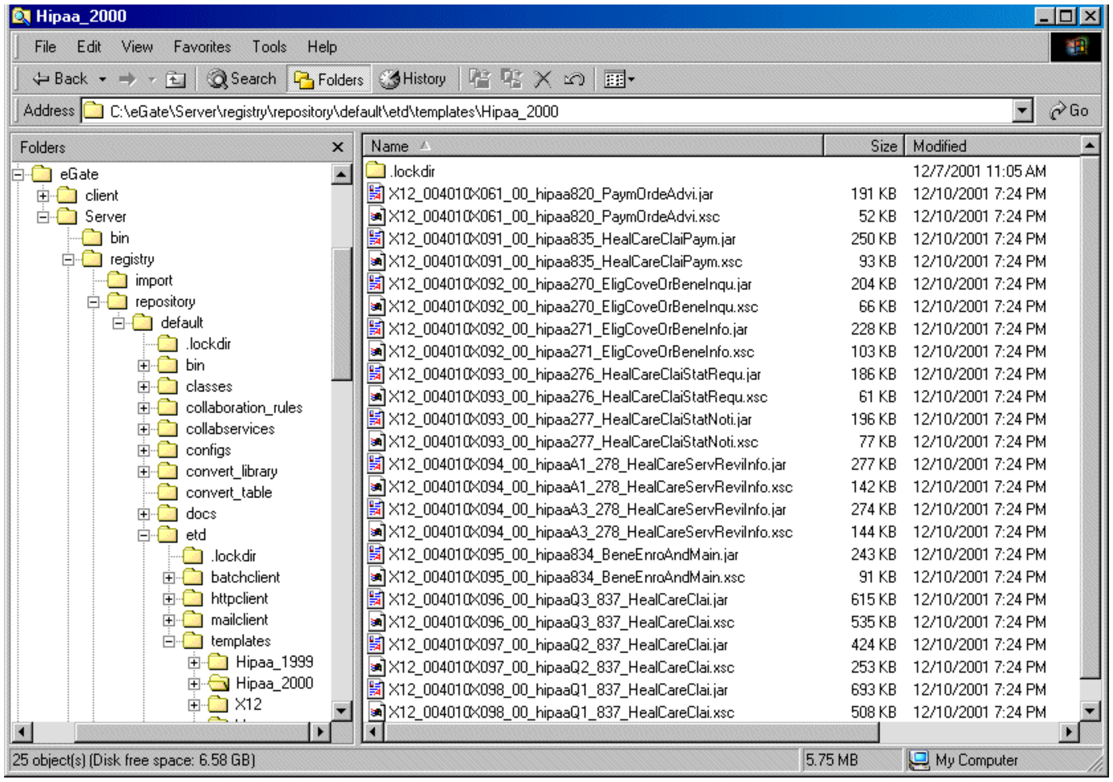
```
.lockdir
X12_004010X061_99_hipaa820_PaymOrdeAdvi.jar
X12_004010X061_99_hipaa820_PaymOrdeAdvi.xsc
X12_004010X091_99_hipaa835_HealCareClaiPaym.jar
X12_004010X091_99_hipaa835_HealCareClaiPaym.xsc
X12_004010X092_99_hipaa270_EligCoveOrBeneInqu.jar
X12_004010X092_99_hipaa270_EligCoveOrBeneInqu.xsc
X12_004010X092_99_hipaa271_EligCoveOrBeneInfo.jar
X12_004010X092_99_hipaa271_EligCoveOrBeneInfo.xsc
X12_004010X093_99_hipaa277_HealCareClaiStatNoti.jar
X12_004010X093_99_hipaa277_HealCareClaiStatNoti.xsc
X12_004010X093_99_hippa276_HealCareClaiStatRequ.jar
X12_004010X093_99_hippa276_HealCareClaiStatRequ.xsc
X12_004010X094_99_hipaaA1_278_HealCareServReviInfo.jar
X12_004010X094_99_hipaaA1_278_HealCareServReviInfo.xsc
X12_004010X094_99_hipaaA3_278_HealCareServReviInfo.jar
X12_004010X094_99_hipaaA3_278_HealCareServReviInfo.xsc
X12_004010X095_99_hipaa834_BeneEnroAndMain.jar
X12_004010X095_99_hipaa834_BeneEnroAndMain.xsc
X12_004010X096_99_hipaaQ3_837_HealCareClai.jar
X12_004010X096_99_hipaaQ3_837_HealCareClai.xsc
X12_004010X097_99_hipaaQ2_837_HealCareClai.jar
X12_004010X097_99_hipaaQ2_837_HealCareClai.xsc
X12_004010X098_99_hipaaQ1_837_HealCareClai.jar
X12_004010X098_99_hipaaQ1_837_HealCareClai.xsc
```

25 object(s) (Disk free space: 5.81 GB) | 7.03 MB | My Computer

3.3.2 HIPAA Files

Figure 3 shows some of the files that are installed for the May 2000 HIPAA transactions (Java).

Figure 3 Transaction Set Structures for May 2000 HIPAA Transactions (Java)



Address: C:\eGate\Server\registry\repository\default\etd\templates\Hipaa_2000

Name	Size	Modified
.lockdir		12/7/2001 11:05 AM
X12_004010X061_00_hipaa820_PaymOrdeAdvi.jar	191 KB	12/10/2001 7:24 PM
X12_004010X061_00_hipaa820_PaymOrdeAdvi.xsc	52 KB	12/10/2001 7:24 PM
X12_004010X091_00_hipaa835_HealCareClaiPaym.jar	250 KB	12/10/2001 7:24 PM
X12_004010X091_00_hipaa835_HealCareClaiPaym.xsc	93 KB	12/10/2001 7:24 PM
X12_004010X092_00_hipaa270_EligCoveOrBeneInqu.jar	204 KB	12/10/2001 7:24 PM
X12_004010X092_00_hipaa270_EligCoveOrBeneInqu.xsc	66 KB	12/10/2001 7:24 PM
X12_004010X092_00_hipaa271_EligCoveOrBeneInfo.jar	228 KB	12/10/2001 7:24 PM
X12_004010X092_00_hipaa271_EligCoveOrBeneInfo.xsc	103 KB	12/10/2001 7:24 PM
X12_004010X093_00_hipaa276_HealCareClaiStatRequ.jar	186 KB	12/10/2001 7:24 PM
X12_004010X093_00_hipaa276_HealCareClaiStatRequ.xsc	61 KB	12/10/2001 7:24 PM
X12_004010X093_00_hipaa277_HealCareClaiStatNoti.jar	196 KB	12/10/2001 7:24 PM
X12_004010X093_00_hipaa277_HealCareClaiStatNoti.xsc	77 KB	12/10/2001 7:24 PM
X12_004010X094_00_hipaaA1_278_HealCareServReviInfo.jar	277 KB	12/10/2001 7:24 PM
X12_004010X094_00_hipaaA1_278_HealCareServReviInfo.xsc	142 KB	12/10/2001 7:24 PM
X12_004010X094_00_hipaaA3_278_HealCareServReviInfo.jar	274 KB	12/10/2001 7:24 PM
X12_004010X094_00_hipaaA3_278_HealCareServReviInfo.xsc	144 KB	12/10/2001 7:24 PM
X12_004010X095_00_hipaa834_BeneEnroAndMain.jar	243 KB	12/10/2001 7:24 PM
X12_004010X095_00_hipaa834_BeneEnroAndMain.xsc	91 KB	12/10/2001 7:24 PM
X12_004010X096_00_hipaaQ3_837_HealCareClai.jar	615 KB	12/10/2001 7:24 PM
X12_004010X096_00_hipaaQ3_837_HealCareClai.xsc	535 KB	12/10/2001 7:24 PM
X12_004010X097_00_hipaaQ2_837_HealCareClai.jar	424 KB	12/10/2001 7:24 PM
X12_004010X097_00_hipaaQ2_837_HealCareClai.xsc	253 KB	12/10/2001 7:24 PM
X12_004010X098_00_hipaaQ1_837_HealCareClai.jar	693 KB	12/10/2001 7:24 PM
X12_004010X098_00_hipaaQ1_837_HealCareClai.xsc	508 KB	12/10/2001 7:24 PM

25 object(s) (Disk free space: 6.58 GB) 5.75 MB

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 will include delimiters for the optional fields.*

3.3.3 File Names

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

HIPAA File Names

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

- X12_ (name of standard followed by underscore)
- 004010X092_ (HIPAA reference number for the transaction—which includes the X12 version—followed by underscore)
- Year designation:
  - 99_ for 1999 files
  - 00_ for 2000 files
- Hipaa_ (indication that this is a HIPAA ETD, followed by underscore)

- 277_ (transaction ID)
- 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_99_hipaa270_EligCoveOrBeneInqu.xsc**

NCPDP File Names

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

- NCPDP_ (name of standard followed by underscore)
- T51_ , Batch_1_0, or Batch11 (version type and number, version type is followed by underscore)
- REQ_ or RESP_ (indicates whether the message is a request or a response)
- Two-character transaction code followed by underscore; for example, E1_ or N3_ (indicates transaction type, such as Eligibility Verification or Information Reporting Rebill)
- 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**.

Chapter 4

Working With the HIPAA X12 ETDs

This chapter provides information on additional features built into the X12 ETDs, and instructions on working with the ETDs and on testing them.

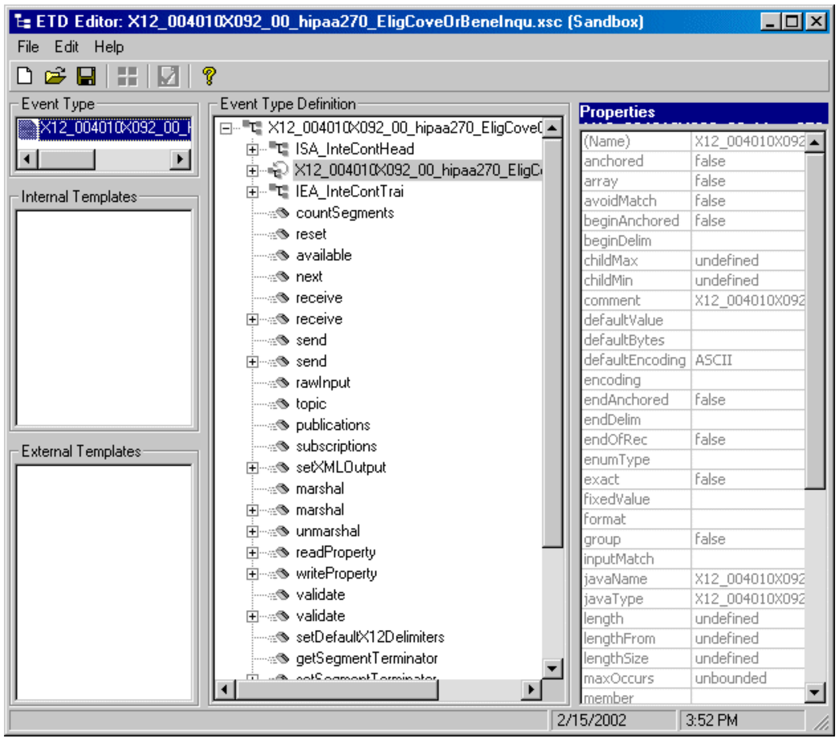
To test that your data is being mapped correctly by the ETD, and that the data is valid based on definitions and business rules, you can run validation within the Collaboration Rules component.

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.

4.1 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



ETD Editor: X12_004010X092_00_hipaa270_EligCoveOrBeneInqu.xsc (Sandbox)

X12_004010X092_00_H

Event Type Definition

```
X12_004010X092_00_hipaa270_EligCove(
    ISA_InteContHead
    X12_004010X092_00_hipaa270_EligC
    IEA_InteContTrai
    countSegments
    reset
    available
    next
    receive
    receive
    send
    send
    rawInput
    topic
    publications
    subscriptions
    setXMLOutput
    marshal
    marshal
    unmarshal
    readProperty
    writeProperty
    validate
    validate
    setDefaultX12Delimiters
    getSegmentTerminator
```

(Name)	X12_004010X092
anchored	false
array	false
avoidMatch	false
beginAnchored	false
beginDelim	
childMax	undefined
childMin	undefined
comment	X12_004010X092
defaultValue	
defaultBytes	
defaultEncoding	ASCII
encoding	
endAnchored	false
endDelim	
endOfRec	false
enumType	
exact	false
fixedValue	
format	
group	false
inputMatch	
javaName	X12_004010X092
javaType	X12_004010X092
length	undefined
lengthFrom	undefined
lengthSize	undefined
maxOccurs	unbounded
member	

2/15/2002 3:52 PM

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**

4.2 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)
- Repetition Separator (version 4020 and later) (default is a plus sign)
- 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.

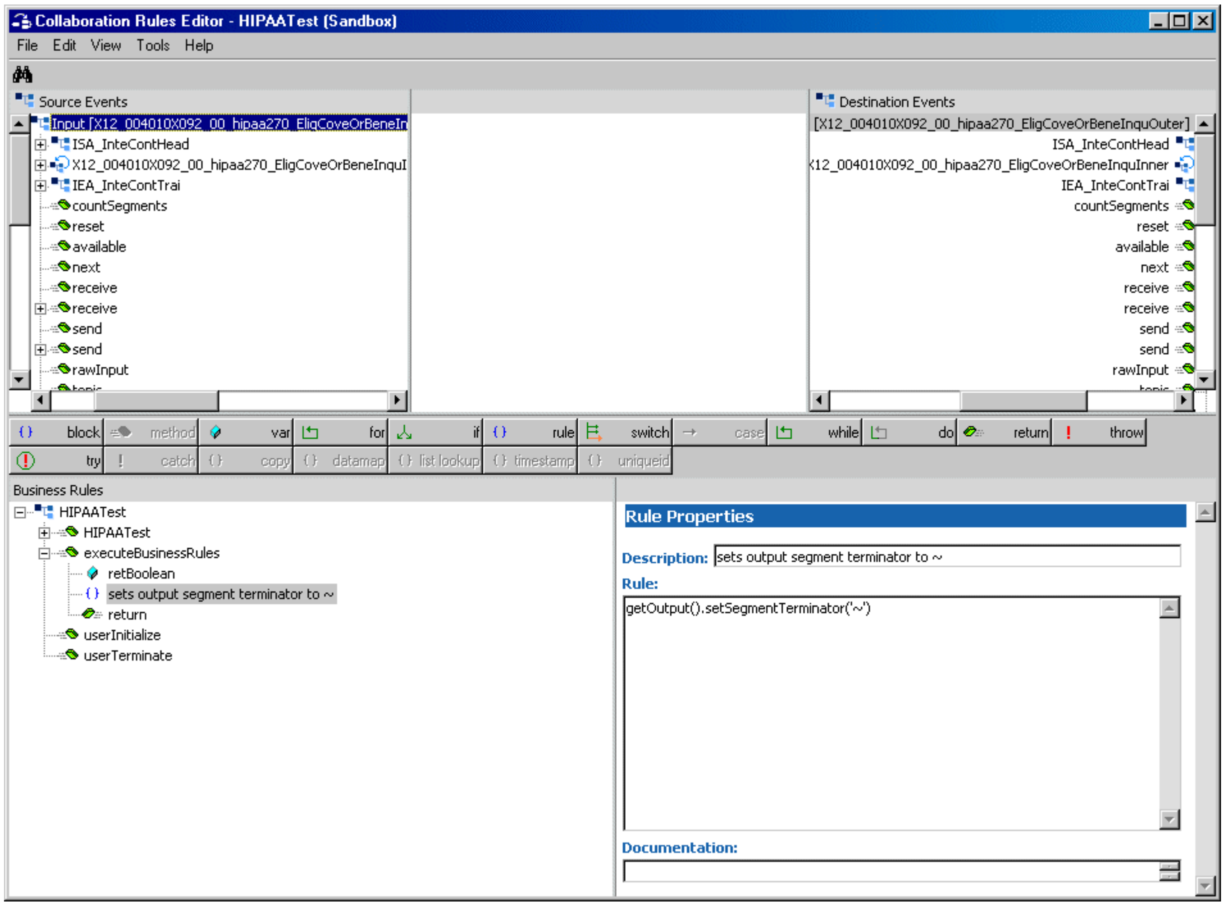
For specific information on the Java methods provided for the getting and setting of delimiters, refer to **"HIPAA ETD Library Java Methods" on page 28**.

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 to the Rule Properties section (an example is shown in Figure 5).

Figure 5 Setting Delimiters in a Collaboration Rules Component


Collaboration Rules Editor - HIPAATest (Sandbox)
File Edit View Tools Help
Source Events
Input [X12_004010X092_00_hipaa270_EligCoveOrBeneIn
ISA_InteContHead
X12_004010X092_00_hipaa270_EligCoveOrBeneInquI
IEA_InteContTrai
countSegments
reset
available
next
receive
receive
send
send
rawInput
Destination Events
[X12_004010X092_00_hipaa270_EligCoveOrBeneInquOuter]
ISA_InteContHead
X12_004010X092_00_hipaa270_EligCoveOrBeneInquInner
IEA_InteContTrai
countSegments
reset
available
next
receive
receive
send
send
rawInput
block method var for if rule switch case while do return throw
try catch copy datamap list lookup timestamp uniqueid
Business Rules
HIPAATest
HIPAATest
executeBusinessRules
retBoolean
sets output segment terminator to ~
return
userInitialize
userTerminate
Rule Properties
Description:
Rule:
Documentation:

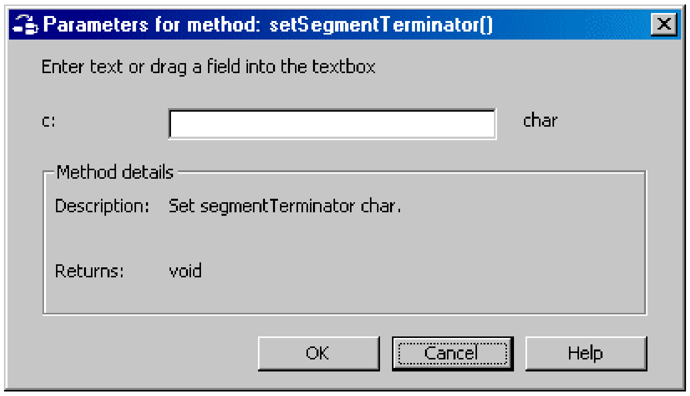
Description: sets output segment terminator to ~

Rule:

```
getOutput().setSegmentTerminator('~')
```

5 The Parameters for Method dialog box appears (see Figure 6).

Figure 6 Parameters for Method Dialog Box

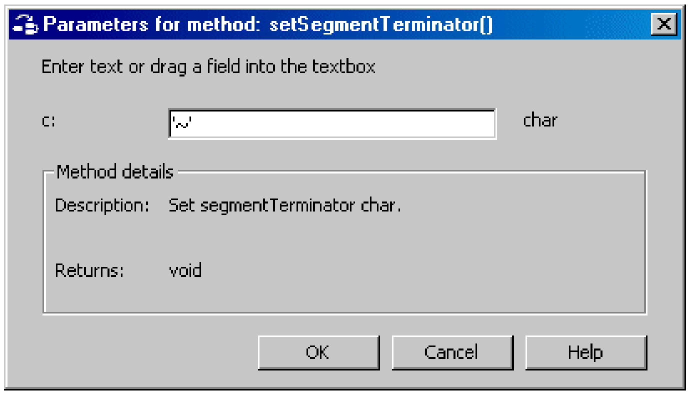

Parameters for method: setSegmentTerminator()
Enter text or drag a field into the textbox
c:
char
Method details
OK
Cancel
Help

Description: Set segmentTerminator char.

Returns: void

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

Figure 7 Parameters for Method Dialog Box Showing Delimiter Value


Parameters for method: setSegmentTerminator()

Enter text or drag a field into the textbox

c: '~' char

Method details

Description: Set segmentTerminator char.

Returns: void

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.*

4.3 Running Validation in the Collaboration Rules Component

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

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

- validate
- validate(boolean)

For more information on these Java methods, refer to **"HIPAA ETD Library Java Methods" on page 28**.

4.3.1 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.

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

It also performs specific validations on the Interchange Group and Functional Group envelopes, and outputs any invalid information into the output string, as follows:

- Checks that the control numbers in the ISA and IEA segments match.
- It checks the number of transactions and verifies that against the transaction count value provided in the GE01 segment of the Functional Group trailer (GE).
- validates the transaction count; checks the number of transactions and checks it against the count provided, says the transaction is invalid if they don't match.

The function returns a string. 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.*

4.4 Alternative Formats: ANSI and XML

All the HIPAA X12 ETDs accept either standard ANSI X12 format or XML format as input, by default; 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 4.5.2 version of the HIPAA X12 ETD Library.

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

4.4.1 XML Format for HIPAA X12

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

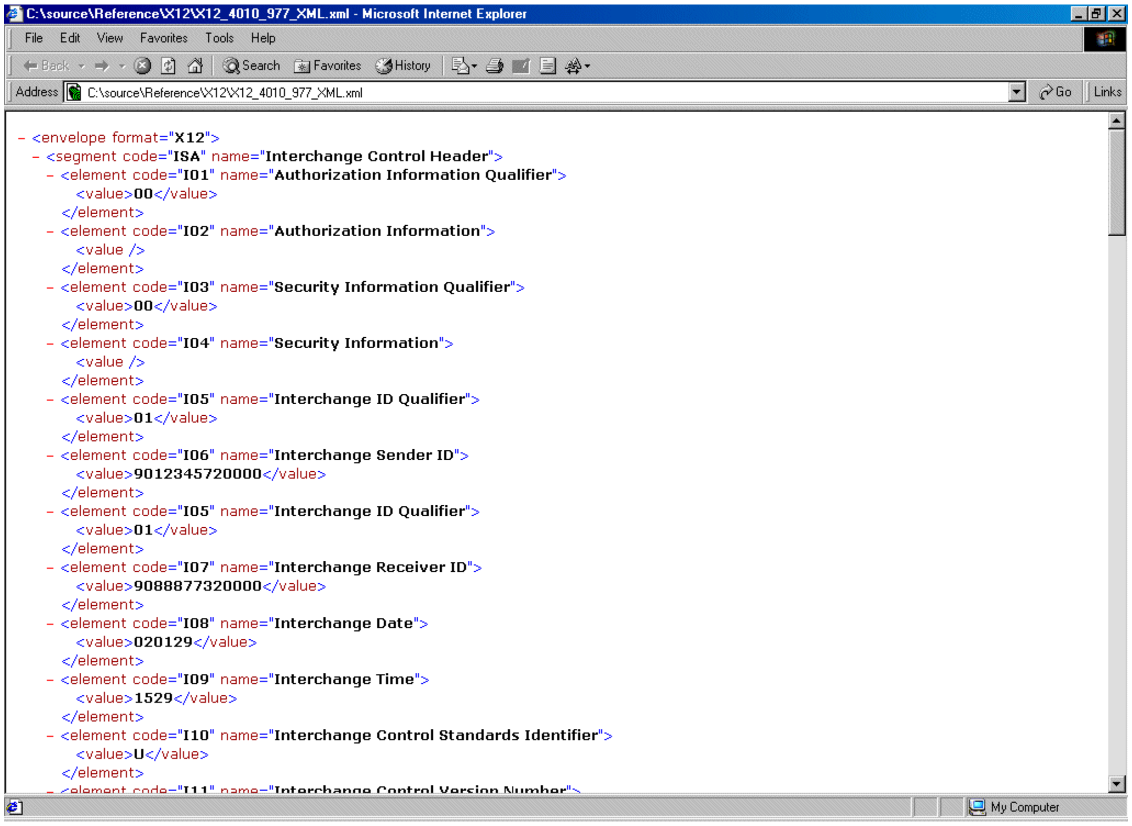
The XML X12 DTD is shown in Figure 8.

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.

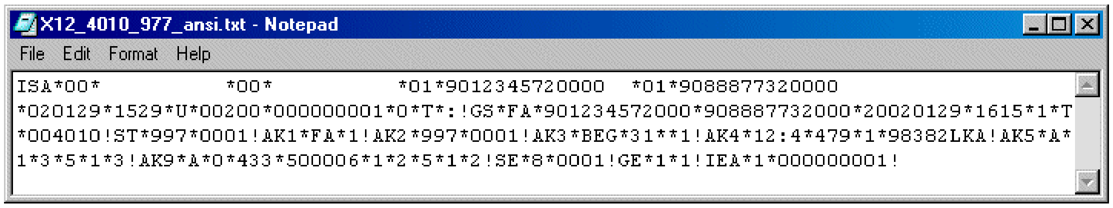
Figure 9 X12 997 Functional Acknowledgment—XML


C:\source\Reference\X12\X12_4010_977_XML.xml - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address C:\source\Reference\X12\X12_4010_977_XML.xml
Go Links
My Computer

```
- <envelope format="X12">
  - <segment code="ISA" name="Interchange Control Header">
    - <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>
    - <element code="I11" name="Interchange Control Version Number">
```

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



```
ISA*00*          *00*          *01*9012345720000  *01*9088877320000
*020129*1529*U*00200*000000001*0*T*:!GS*FA*901234572000*908887732000*20020129*1615*1*T
*004010!ST*997*0001!AK1*FA*1!AK2*997*0001!AK3*BEG*31**1!AK4*12:4*479*1*98382LKA!AK5*A*
1*3*5*1*3!AK9*A*0*433*500006*1*2*5*1*2!SE*8*0001!GE*1*1!IEA*1*000000001!
```

4.4.2 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.

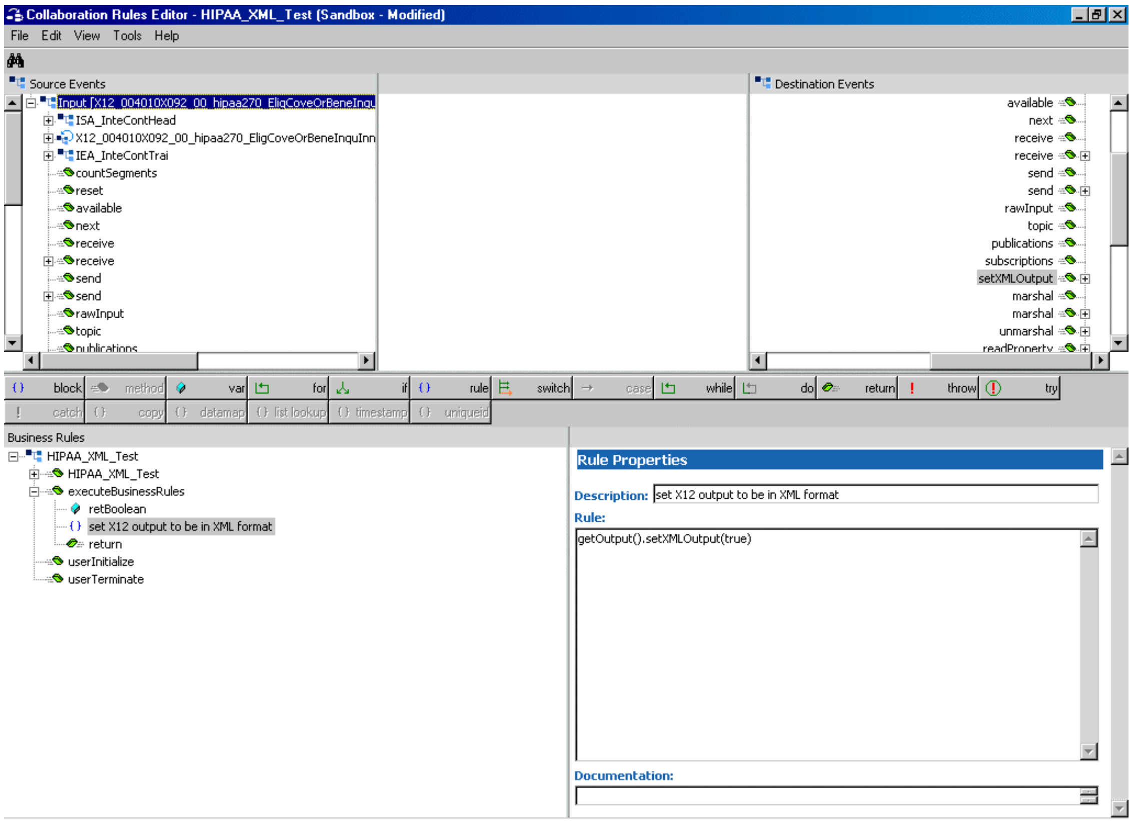
If you want to set the Collaboration to output XML format, use one of the following two new Java methods:

- 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



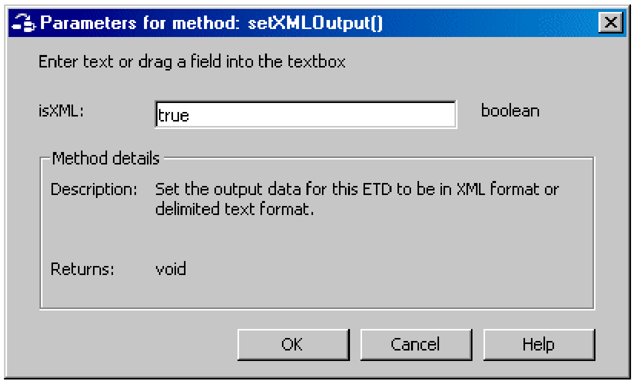
Description: set X12 output to be in XML format

Rule:

```
getOutput().setXMLOutput(true)
```

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

Figure 12 Specifying the Parameter for setXMLOutput ( )


Parameters for method: setXMLOutput()
Enter text or drag a field into the textbox
Method details
OK
Cancel
Help

isXML: true boolean

Description: Set the output data for this ETD to be in XML format or delimited text format.

Returns: void

4.5 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. However, certain differences, 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 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. These methods allow you to get 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:

- **SetDefaultX12Delimiters** on page 28
- **getSegmentTerminator** on page 29
- **setSegmentTerminator** on page 30
- **getElementSeparator** on page 30
- **setElementSeparator** on page 31
- **getSubelementSeparator** on page 31
- **setSubelementSeparator** on page 32
- **getRepetitionSeparator** on page 33
- **setRepetitionSeparator** on page 33

The HIPAA ETD Library also includes the following custom Java methods for testing the validation Collaboration:

- **validate (no parameters)** on page 34
- **validate (boolean parameter)** on page 34

In addition, the library includes the following functions for setting the output of a Collaboration to XML:

- **setXMLOutput (boolean isXML)** on page 35
- **marshal (boolean parameter)** on page 36

SetDefaultX12Delimiters

Description

Sets the default X12 delimiters.

Syntax

```
public void setDefaultX12Delimiters()
```

Parameters

None.

setDefaultX12Delimiters Constants

None.

Returns

void (none).

Throws

None.

Examples

```
com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter myETD=new
 com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter();
......
......
myETD.setDefaultX12Delimiters();
```

getSegmentTerminator

Description

Gets the segmentTerminator character.

Syntax

```
public char getSegmentTerminator()
```

Parameters

None.

getSegmentTerminator Constants

None.

Returns

char
Returns the segment terminator character.

Throws

None.

Examples

```
com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter myETD=new
 com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter();
......
......
char segTerm=myETD.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.

setSegmentTerminator Constants

None.

Returns

void (none).

Throws

None.

Examples

```
com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter myETD=new
 com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter();
......
......
char c='~';
myETD.setSegmentTerminator(c);
```

getElementSeparator

Description

Gets the elementSeparator character.

Syntax

```
public char getElementSeparator()
```

Parameters

None.

getElementSeparator Constants

None.

Returns

char
Returns the element separator character.

Throws

None.

Examples

```
com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter myETD=new
 com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter();
......
......
char elmSep=myETD.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.

setElementSeparator Constants

None.

Returns

void (none).

Throws

None.

Examples

```
com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter myETD=new
 com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter();
......
......
char c='+';
myETD.setElementSeparator(c);
```

getSubelementSeparator

Description

Gets the subelementSeparator character.

Syntax

```
public char getSubelementSeparator()
```

Parameters

None.

getSubelementSeparator Constants

None.

Returns

char

Returns the getSubelement character.

Throws

None.

Examples

```
com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter myETD=new
 com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter();
......
......
char subeleSep=myETD.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.

setSubelementSeparator Constants

None.

Returns

void (none).

Throws

None.

Examples

```
com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter myETD=new
 com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter();
......
......
char c=':';
myETD.setSubelementSeparator(c);
```

getRepetitionSeparator

Description

Gets the RepetitionSeparator character.

Syntax

```
public char getRepetitionSeparator()
```

Parameters

None.

getRepetitionSeparator Constants

None.

Returns

char

Returns the getRepetitionSeparator character.

Throws

None.

Examples

```
com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter myETD=new
 com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter();
......
......
char repSep=myETD.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.

setRepetitionSeparator Constants

None.

Returns

void (none).

Throws

None.

Examples

```
com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter myETD=new
 com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter();
......
......
char c='*';
myETD.setRepetitionSeparator(c);
```

validate (no parameters)

Description

Validates the ETD content in memory.

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.

Syntax

```
public String validate()
```

Parameters

None.

validate 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.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter myETD=new
 com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter();
......
......
string msg=myETD.validate();
```

validate (boolean parameter)

Description

Validates the ETD content, either immediately after unmarshaling or in memory.

When used with the parameter set to false, this method works in the same way as validate (with no parameters).

However, when the parameter is set to true, this method can be used to validate length information in the input data file.

For example, if the ETD expects a six-digit date and the input data provides an eight-digit date, this method outputs a string such as the following:

```
X12_004010X092_00_hipaa270_EligCoveOrBeneInquOuter.X12_004010X092_00_
hipaa270_EligCoveOrBeneInquInner[0].GS_FuncGrouHead.E29_4_GrouDate:
Its length of [8] is more than required max of [6].
```

Syntax

```
public String validate(boolean original)
```

Parameters

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

validate 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.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter myETD=new
 com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter();
......
......
string msg=myETD.validate(true);
```

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.

setXMLOutput Constants

None.

Returns

void (none).

Throws

None.

Examples

```
com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter myETD=new
 com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter();
......
......
myETD.setXMLOutput(true);
```

marshal (boolean parameter)

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.

marshal Constants

None.

Returns

byte []

The output in byte array format.

Throws

None.

Examples

```
com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter myETD=new
 com.stc.x12.X12_004010X098_00_hipaaQ1_837_HealCareClaiOuter();
......
......
byte[] output=myETD.marshal(true);
```

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 e*Gate 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 41**.

A.2.3 Loops

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

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

A.2.4 Delimiters

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

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

The default delimiters used by the SeeBeyond HIPAA ETD Library are the same as those recommended by the industry-specific implementation guides. These delimiters are shown in Table 4.

Table 4 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

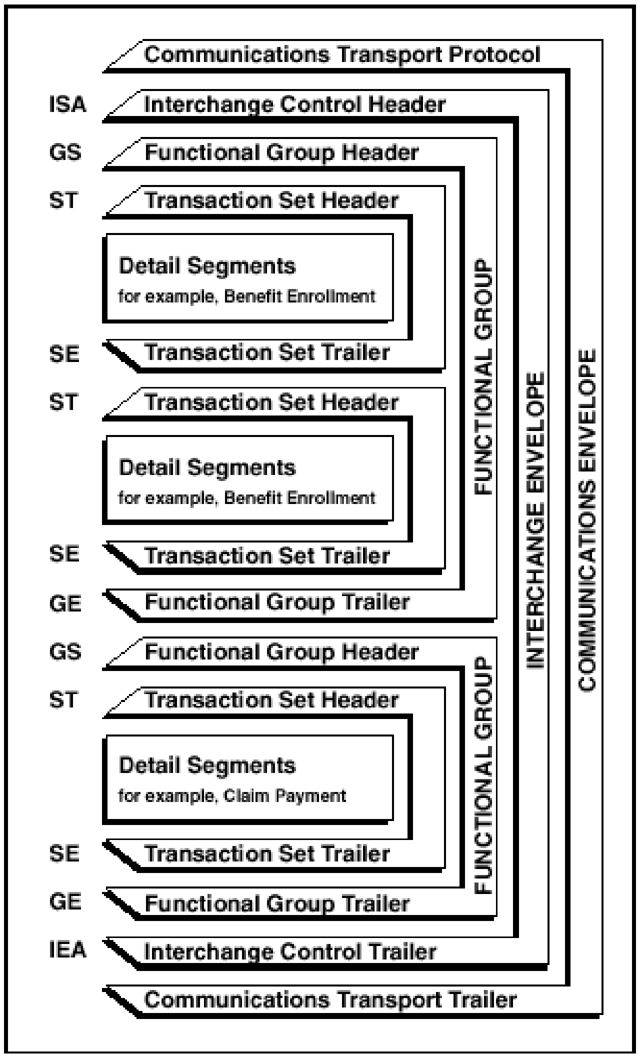
The rules applying to the structure of an X12 envelope are very strict, to ensure the integrity of the data and the efficiency of the information exchange.

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

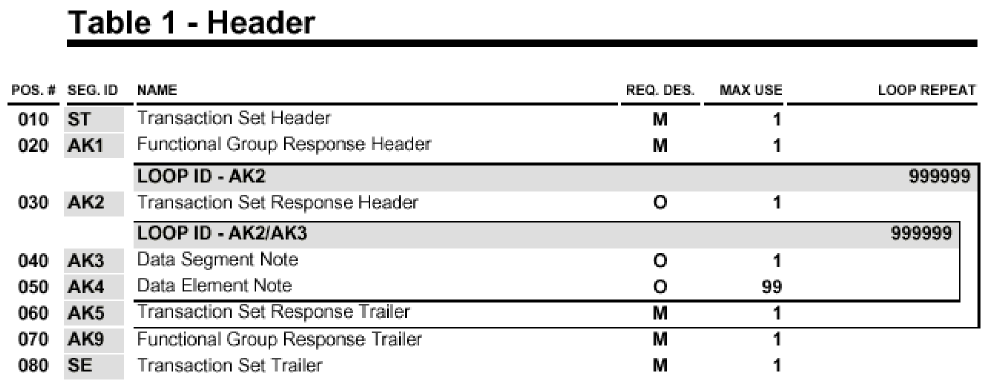
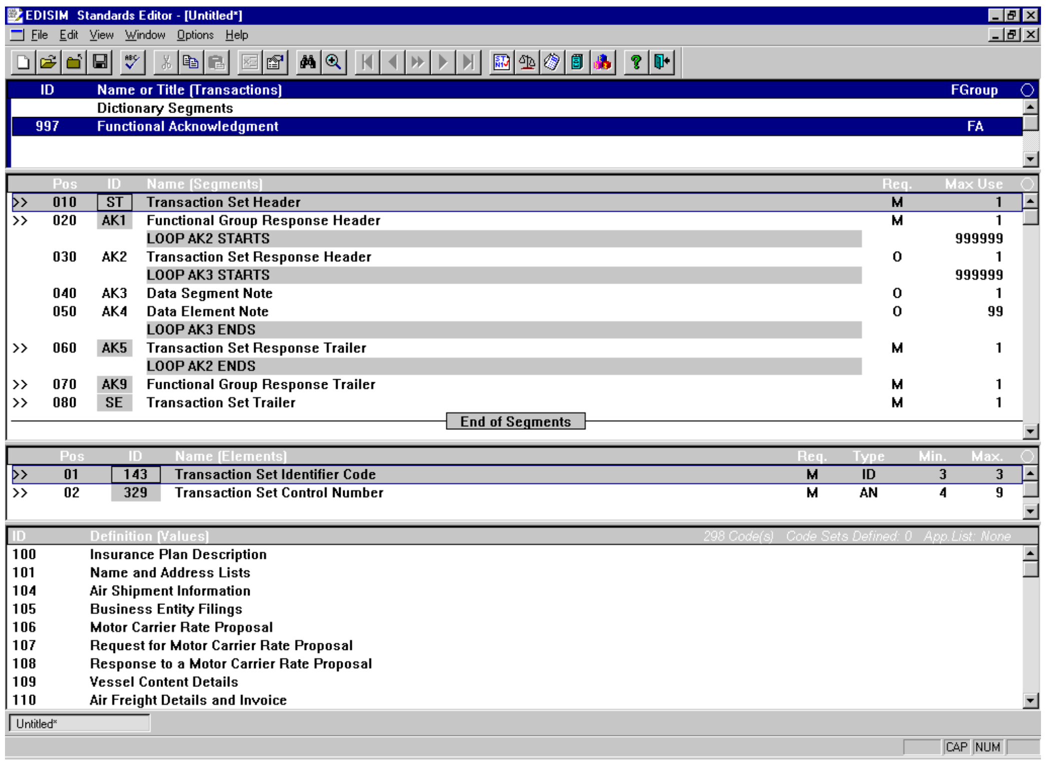


Table 1 - Header

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 a third-party implementation guide editor. You can use the third-party tool to edit the transaction and export it in the EDI Standard Exchange Format (SEF).

Figure 15 X12 997 Open in Third-Party Tool



ID	Name or Title (Transactions)	FGroup
	Dictionary Segments	
997	Functional Acknowledgment	FA

	Pos	ID	Name (Segments)	Req.	Max Use
>>	010	ST	Transaction Set Header	M	1
>>	020	AK1	Functional Group Response Header	M	1
			LOOP AK2 STARTS		999999
	030	AK2	Transaction Set Response Header	O	1
			LOOP AK3 STARTS		999999
	040	AK3	Data Segment Note	O	1
	050	AK4	Data Element Note	O	99
			LOOP AK3 ENDS		
>>	060	AK5	Transaction Set Response Trailer	M	1
			LOOP AK2 ENDS		
>>	070	AK9	Functional Group Response Trailer	M	1
>>	080	SE	Transaction Set Trailer	M	1

End of Segments

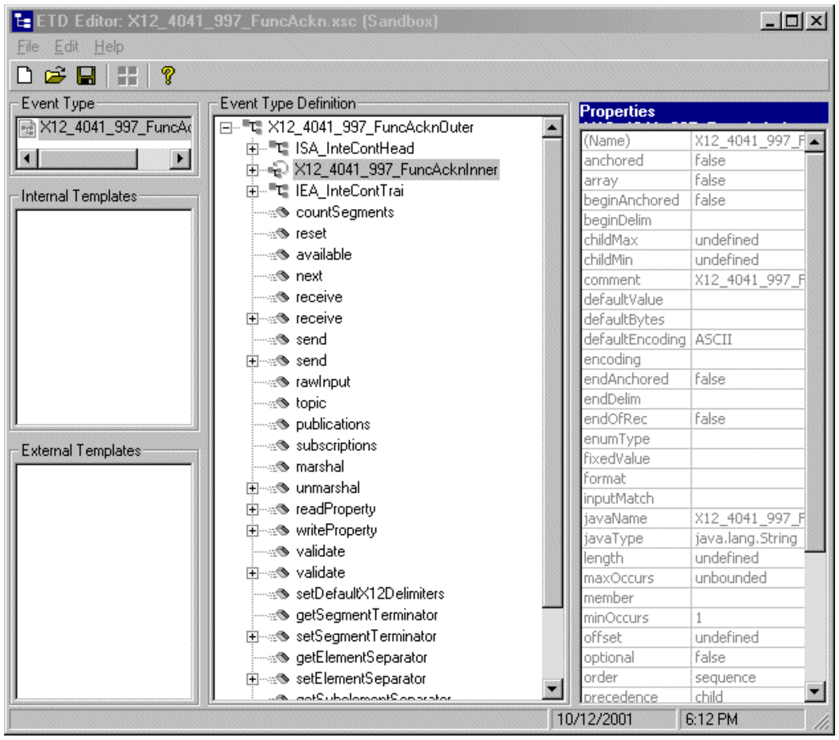
	Pos	ID	Name (Elements)	Req.	Type	Min.	Max.
>>	01	143	Transaction Set Identifier Code	M	ID	3	3
>>	02	329	Transaction Set Control Number	M	AN	4	9

298 Code(s) Code Sets Defined: 0 App.List: None

ID	Definition (Values)
100	Insurance Plan Description
101	Name and Address Lists
104	Air Shipment Information
105	Business Entity Filings
106	Motor Carrier Rate Proposal
107	Request for Motor Carrier Rate Proposal
108	Response to a Motor Carrier Rate Proposal
109	Vessel Content Details
110	Air Freight Details and Invoice

Figure 16 shows the same transaction as viewed in the Java ETD Editor.

Figure 16 X12 997 Viewed in Java ETD Editor



X12_4041_997_FuncAc

```
X12_4041_997_FuncAcknOuter
  ISA_InteContHead
  X12_4041_997_FuncAcknInner
  IEA_InteContTrai
  countSegments
  reset
  available
  next
  receive
  receive
  send
  send
  rawInput
  topic
  publications
  subscriptions
  marshal
  unmarshal
  readProperty
  writeProperty
  validate
  validate
  setDefaultX12Delimiters
  getSegmentTerminator
  setSegmentTerminator
  getElementSeparator
  setElementSeparator
```

(Name)	X12_4041_997_F
anchored	false
array	false
beginAnchored	false
beginDelim	
childMax	undefined
childMin	undefined
comment	X12_4041_997_F
defaultValue	
defaultBytes	
defaultEncoding	ASCII
encoding	
endAnchored	false
endDelim	
endOfRec	false
enumType	
fixedValue	
format	
inputMatch	
javaName	X12_4041_997_F
javaType	java.lang.String
length	undefined
maxOccurs	unbounded
member	
minOccurs	1
offset	undefined
optional	false
order	sequence
precedence	child

10/12/2001 6:12 PM

A.3.1 Transaction Set (ST/SE)

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

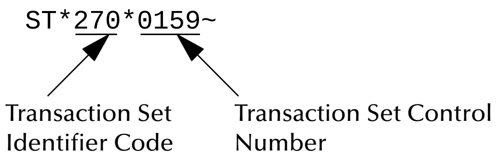
- A transaction set header
- A transaction set footer
- A single message, enveloped within the header and footer

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

The transaction is comprised 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.

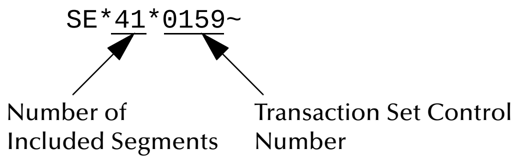
Figure 17 Example of a Transaction Set Header (ST)



```
ST*270*0159~
```

Transaction Set Identifier Code
Transaction Set Control Number

Figure 18 Example of a Transaction Set Trailer (SE)



```
SE*41*0159~
```

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 19 Example of a Functional Group Header (GS)

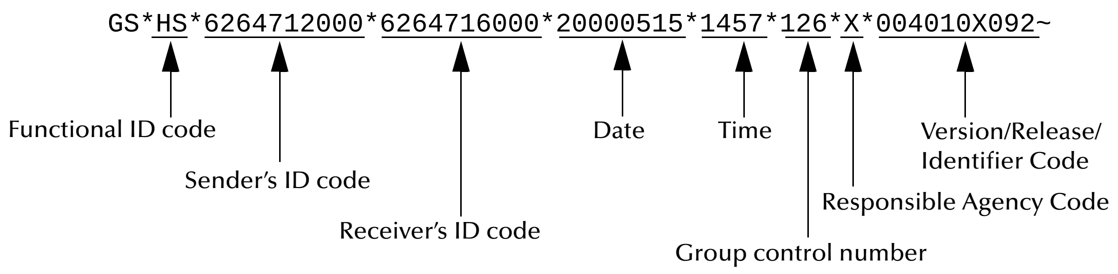
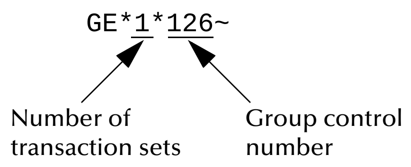


Figure 20 Example of a Functional Group Trailer (GE)



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.

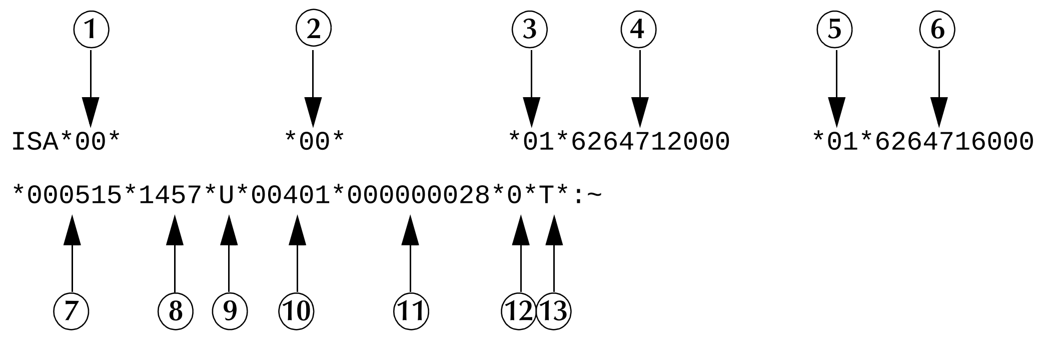
As well as enveloping one or more functional groups, the interchange header and trailer segments include the following information:

- 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

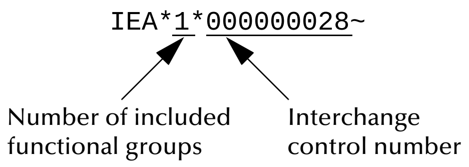
Figure 21 Example of an Interchange Header (ISA)


1
2
3
4
5
6
ISA*00*   *00*   *01*6264712000   *01*6264716000
*000515*1457*U*00401*000000028*0*T*:~
7
8
9
10
11
12
13

Interchange Header Segments from Figure 21:

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

Figure 22 Example of an Interchange Trailer (IEA)


IEA*1*000000028~
Number of included functional groups
Interchange control number

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 footer, and must be unique within a Functional Group (GS-GE).*

A.4 Security

EDI-INT is an international standard for secure EDI transmissions, both X12 and UN/EDIFACT. It is emerging as a widespread EDI security standard.

EDI for the Internet is an international standard for secure EDI transmissions, both X12 and UN/EDIFACT, published by the Internet Engineering Task Force (IETF).

EDI-INT documentation specifies how to use internet mail protocols to transport EDI documents, using either PGP/MIME or S/MIME. It also discusses the implementation of encryption, digital signature, integrity and signed-receipt for MIME objects transported over SMTP, HTTP or FTP.

For additional information visit:

http://www.ietf.cnri.reston.va.us/ids.by.wg/ediint.html

The SeeBeyond eBusiness Integration Suite provides the following security-related features in various products and add-ons:

- The e*Gate Secure Messaging Extension
- HTTPS e*Way
- e*Xchange Security Manager

A.5 Acknowledgment Types

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

A.5.1 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.5.2 997, 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.5.3 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.6 Key Parts of EDI Processing Logic

The five key parts of EDI processing logic are listed in Table 5. The table describes each term, and lists its language analogy along with its associated e*Gate Collaboration scripts.

Table 5 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.6.1 Structures

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

A.6.2 Validations, Translations, Enveloping, Acknowledgments

e*Gate does not include any pre-built validations, transformations, enveloping, 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.

However, installation of the e*Xchange Partner Manager includes a set of custom Monk validations for HIPAA transactions.

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

A.6.3 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.7 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, SeeBeyond has no control over these sites. If you find the links are no longer correct, use a search engine to search for **X12**.*

Index