SeeBeyond ICAN Suite

HIPAA OTD Library User's Guide

Release 5.0.2

SeeBeyond Proprietary and Confidential

The information contained in this document is subject to change and is updated periodically to reflect changes to the applicable software. Although every effort has been made to ensure the accuracy of this document, SeeBeyond Technology Corporation (SeeBeyond) assumes no responsibility for any errors that may appear herein. The software described in this document is furnished under a License Agreement and may be used or copied only in accordance with the terms of such License Agreement. Printing, copying, or reproducing this document in any fashion is prohibited except in accordance with the License Agreement. The contents of this document are designated as being confidential and proprietary; are considered to be trade secrets of SeeBeyond; and may be used only in accordance with the License Agreement, as protected and enforceable by law. SeeBeyond assumes no responsibility for the use or reliability of its software on platforms that are not supported by SeeBeyond.

SeeBeyond, e*Gate, e*Way, and e*Xchange are the registered trademarks of SeeBeyond Technology Corporation in the United States and/or select foreign countries. The SeeBeyond logo, SeeBeyond Integrated Composite Application Network Suite, eGate, eWay, eInsight, eVision, eXchange, eView, eIndex, eTL, ePortal, eBAM, and e*Insight are trademarks of SeeBeyond Technology Corporation. The absence of a trademark from this list does not constitute a waiver of SeeBeyond Technology Corporation's intellectual property rights concerning that trademark. This document may contain references to other company, brand, and product names. These company, brand, and product names are used herein for identification purposes only and may be the trademarks of their respective owners.

© 2005 by SeeBeyond Technology Corporation. All Rights Reserved. This work is protected as an unpublished work under the copyright laws.

This work is confidential and proprietary information of SeeBeyond and must be maintained in strict confidence.

Version 20050401063314.

Contents

List of Figures 6

Chapter 1

Introduction 7

About This Document 7
What's In This Document? 7
Scope 8
Intended Audience 8
Document Conventions 8
Screenshots 8
Related Documents 8

References 9

SeeBeyond Web Site 9

SeeBeyond Documentation Feedback 9

Chapter 2

Overview of the HIPAA OTD Library 10

About the HIPAA OTD Library 10

SEF File Support 11

HIPAA Validation Support 11

On Demand Parsing 12

Alternative Formats: ANSI and XML 12
XML Format for X12 13
XML X12 DTD 13
Sample XML X12 Output 13
Sample of ANSI Output 14

Errors and Exceptions 14

Chapter 3

Installing the HIPAA OTDs 15

System Requirements 15

Supported Operating Systems 15
Installing the HIPAA OTD Library 16
Increasing the Enterprise Designer Heap Size 17
Resolving Memory Errors at Enterprise Designer Startup 17

Chapter 4
Using HIPAA OTDs 18
Displaying HIPAA OTDs 18
Building HIPAA OTD Collaborations 19
Customizing the HIPAA OTDs 21
Creating HIPAA OTDs from SEF Files 23
Possible Differences in Output When Using Pass-Through 25

Chapter 5
Java Methods for HIPAA OTDs 26
Get and Set Methods 26
Setting Delimiters 27
Incoming Message 27
Outgoing Message 28
Available Methods 28
check 28
checkAll 28
clone 29
count*xxx* 29
countLoop*xxx* 29
getxxx 30
getAllErrors 30
getElementSeparator 30
getFGValidationResult 30
getICValidationResult 31
getInputSource 31
getLoopxxx 31
getMaxDataError 32
getMaxFreedSegsComsNum 32
getMaxParsedSegsComsNum 32
getMsgValidationResult 32
getRepetitionSeparator 33
getSegmentCount 33
getSegmentTerminator 33
getSubelementSeparator 34
getTSValidationResult 34
getUnmarshalError 34
getXmlOutput 34
hasxxx 35
has*Loopxxx* 35
isUnmarshalComplete 35
marshal 36

marshalToBytes 36
marshalToString 36
performValidation 36
reset 37
set*xxx* 37
setDefaultX12Delimiters 37
setElementSeparator 38
setLoopxxx 38
setMaxDataError 38
setMaxFreedSegsComsNum 39
setMaxParsedSegsComsNum 39
setRepetitionSeparator 39
setSegmentTerminator 39
setSubelementSeparator 40
setXmlOutput 40
unmarshal 40
unmarshalFromBytes 41
unmarshalFromString 41

Appendix A

X12OTDErrors Schema File and Sample XML 42

Contents of the X12OTDErrors.xsd File 42

Sample of Validation Output XML 43

Index 45

List of Figures

Figure 1	Increasing Enterprise Designer Heap Size	17
Figure 2	OTDs for the 2000 Standard	19
Figure 3	Selecting the Web Service	20
Figure 4	Adding OTDs to the Collaboration	21
Figure 5	Saving HIPAA OTD SEF Files	22
Figure 6	Creating HIPAA OTDs	23
Figure 7	Selecting the SEF File	24
Figure 8	Selecting the OTD Options	24

Chapter 1

Introduction

This chapter provides an overview of the this user's guide, including its contents and writing conventions.

What's in This Chapter

- **About This Document** on page 7
- **Related Documents** on page 8
- **References** on page 9
- **SeeBeyond Web Site** on page 9
- **SeeBeyond Documentation Feedback** on page 9

1.1 About This Document

The sections below provide information about this document, such as an overview of its contents, scope, and intended audience.

1.1.1 What's In This Document?

This guide contains the following information:

- **Chapter 1, "Introduction"**, provides a preview of this document, its purpose, scope, and organization.
- **Chapter 2, "Overview of the HIPAA OTD Library"**, provides an overview of the HIPAA OTD Library as well as its support for SEF file versions, and validation.
- **Chapter 3, "Installing the HIPAA OTDs"**, describes how to install HIPAA OTDs, the SEF OTD wizard, and the HIPAA OTD Library documentation.
- **Chapter 4, "Using HIPAA OTDs"**, describes how to display and customize OTDs, and how to build Collaborations with HIPAA OTDs.
- **Chapter 5, "Java Methods for HIPAA OTDs"**, provides the syntax for the Java methods provided with the HIPAA OTDs.
- **Appendix A, "X12OTDErrors Schema File and Sample XML"**, provides the X12OTDErrors schema file and a sample validation output XML.

1.1.2 Scope

This document describes the HIPAA OTD library, how to install it, and how to use it with eGate Integrator. For detailed information about eGate-specific procedures, refer to the *eGate Integrator User's Guide*. If you are using the OTD library with eXchange, refer to the *eXchange Integrator User's Guide* for eXchange-specific procedures.

1.1.3 Intended Audience

This document provides information for those who are designing, deploying, and managing ICAN Projects that use HIPAA OTDs. This document assumes that you are familiar with eGate-specific procedures.

1.1.4 Document Conventions

The following conventions are observed throughout this document.

Table 1 Document Conventions

Text	Convention	Example
Names of buttons, files, icons, parameters, variables, methods, menus, and objects	**Bold** text	▪ Click **OK** to save and close. ▪ From the **File** menu, select **Exit**. ▪ Select the **logicalhost.exe** file. ▪ Enter the **timeout** value. ▪ Use the **getClassName()** method. ▪ Configure the **Inbound** File eWay.
Command line arguments, code samples	`Fixed` font. Variables are shown in ***`bold italic`***.	`bootstrap -p` ***`password`***
Hypertext links	**Blue** text	See **Document Conventions** on page 8
Hypertext links for Web addresses (URLs) or email addresses	**Blue underlined** text	**http://www.seebeyond.com** **docfeedback@seebeyond.com**

1.1.5 Screenshots

Depending on what products you have installed, and how they are configured, the screenshots in this document may differ from what you see on your system.

1.1.6 Related Documents

The following SeeBeyond documents provide additional information about the SeeBeyond ICAN Suite:

- *SeeBeyond ICAN Suite Installation Guide*
- *eGate Integrator User's Guide*
- *eGate Integrator JMS Reference Guide*

- *eGate Integrator System Administrator Guide*
- *eGate Integrator Deployment Guide*
- *eXchange Integrator User's Guide*
- *eXchange Integrator Designer's Guide*
- *eInsight Business Process Manager User's Guide*

1.2 References

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

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

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

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

1.3 SeeBeyond Web Site

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

http://www.seebeyond.com

1.4 SeeBeyond Documentation Feedback

We appreciate your feedback. Please send any comments or suggestions regarding this document to:

docfeedback@seebeyond.com

Chapter 2

Overview of the HIPAA OTD Library

This chapter provides an overview of the HIPAA OTD Library as well as its support for SEF file versions, and validation.

What's in This Chapter

- **About the HIPAA OTD Library** on page 10
- **SEF File Support** on page 11
- **HIPAA Validation Support** on page 11
- **On Demand Parsing** on page 12
- **Alternative Formats: ANSI and XML** on page 12
- **Errors and Exceptions** on page 14

2.1 About the HIPAA OTD Library

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.

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

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

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

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

The HIPAA OTD Library provides OTDs for the 2000 Standard and 2000 Addenda HIPAA messages.

HIPAA messages have a message structure, which indicates how data elements are organized and related to each other for a particular EDI transaction. In the ICAN Suite, message structures are defined as OTDs. Each OTD consists of the following:

- Physical hierarchy

  The predefined way in which envelopes, segments, and data elements are organized to describe a particular HIPAA 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 HIPAA OTD Library provides HIPAA OTDs that you can use to build ICAN Projects for interfacing with HIPAA systems. You can use the OTDs standalone with eGate Integrator or in combination with eXchange Integrator and eGate Integrator.

2.2 SEF File Support

The HIPAA OTD Library support SEF versions 1.5 and 1.6 when the SEF OTD wizard is used to build custom OTDs. For more information about the SEF OTD wizard, refer to **"Creating HIPAA OTDs from SEF Files" on page 23**.

The SEF OTD wizard does not handle the following information and sections:

- In the .SEMREFS section, semantic rules with its type of the "exit routine" are ignored as per SEF specification. An exit routine specifies an external routine (such as a COM-enabled server program supporting OLE automation) to run for translators or EDI data analyzers.
- The .TEXT sections (including subsections such as .TEXT,SETS, .TEXT,SEGS, .TEXT,COMS, .TEXT,ELMS, .TEXT,SEGS) are ignored due to the fact that these sections store information about changes in a standard's text, such as notes, comments, names, purposes, descriptions, titles, semantic notes, explanations, and definitions.

2.3 HIPAA Validation Support

Within each HIPAA OTD are Java methods and Java bean nodes for handling validation (see **"performValidation" on page 36**). No pre-built translations are supplied with the OTD libraries; these can be built in the Java Collaboration Editor.

HIPAA OTDs have validations and translations, but a validation does not generate an acknowledgment transaction. Instead, it generates a string.

The output string of the validation (see **"check" on page 28** and **"checkAll" on page 28**) is in XML format conforming to the **X12OTDErrors.xsd** file. Refer to **"Contents of the X12OTDErrors.xsd File" on page 42** for more information. For a sample of the validation output XML, refer to **"Sample of Validation Output XML" on page 43**.

2.4 On Demand Parsing

For performance enhancement reasons, the **unmarshal()** method does not unmarshal the entire message to the leaf element and subelement level. Instead, it does the following:

- Unmarshals the incoming message at the segment and composite level. In other words, the OTD checks for all relevant segments and composites and reports any missing or extra segments or composites.
- Reports excess trailing delimiter for elements and composites.

This is also referred to as "parse on demand," meaning that elements within a segment or composite are not unmarshaled until an element in that segment or composite is accessed by a **get*xxx*()** method invoked in a Collaboration or during marshaling. The OTD may assign unmarshaled segments and composites to a pool that is ready to be freed from memory by the Java Virtual Machine (JVM). Once these segments or composites are freed from memory, they become unparsed. If the element within segment or composite is accessed again, the OTD reparses the segment or composite.

By default, HIPAA OTDs set no limit of parsed segments or composites held in memory. You can specify a limit for parsed and freed segments or composites by using the following methods at the OTD root levels:

- setMaxParsedSegsComsNum() method (**"setMaxParsedSegsComsNum" on page 39**)
- setMaxFreedSegsComsNum() method (**"setMaxFreedSegsComsNum" on page 39**)

You can use these methods to set and control the runtime memory use of the unmarshaling process.

2.5 Alternative Formats: ANSI and XML

The HIPAA OTDs accept either standard ANSI X12 format or XML format as input, by default; you do not need to change the existing Business Processes or Collaborations to specify the input format (standard data is ANSI X12 or XML).

By default, the OTD output is ANSI. To change the output to XML, use the *setXmlOutput (boolean)* method. For information, refer to **"setXmlOutput" on page 40**.

To verify whether the output is XML, use the *getXmlOutput()* method. For information, refer to **"getXmlOutput" on page 34**.

2.5.1. XML Format for X12

Because there is XML standard for X12, the HIPAA OTD Library uses Open Business Objects for EDI (OBOE) as the XML format for X12.

XML X12 DTD

The XML X12 DTD is as follows:

```
<!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>
```

Sample XML X12 Output

Below is an excerpt of the XML X12 output for the 4010 850 transaction.

```
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>011001</value>
    </element>
    <element code="I09" name="Interchange Time">
      <value>1718</value>
    </element>
    <element code="I10" name="Interchange Control Standards Identifier">
      <value>U</value>
    </element>
    <element code="I11" name="Interchange Control Version Number">
      <value>00200</value>
    </element>
    <element code="I12" name="Interchange Control Number">
      <value>000000001</value>
    </element>
    <element code="I13" name="Acknowledgment Requested">
      <value>0</value>
    </element>
    <element code="I14" name="Usage Indicator">
      <value>T</value>
    </element>
    <element code="I15" name="Component Element Separator">
      <value>^</value>
    </element>
  </segment>
```

Sample of ANSI Output

Below is an excerpt of the same transaction in ANSI format:

```
ISA*00*          *00*          *01*9012345720000  *01*9088877320000
*011001*1718*U*00200*000000001*0*T*:~GS*PO*901234572000*908887732000*2
0011001*1615*1*T*004010~ST*850*0001~BEG*01*BK*99AKDF9DAL393*3948392019
3843*20011001*AN3920943*AC*IBM*02*AE*02*BA~CUR*AC*USA*.2939*SE*USA*IMF
*002*20011001*0718*021*20011001*1952*038*20011001*1615*002*20011001*07
18*021*20011001*1952~REF*AB*3920394930203*GENERAL
PURPOSE*BT:12345678900987654321768958473:CM:500:AB:3920394930203~PER*A
C*ARTHUR JONES*TE*(614)555-1212*TE*(614)555-1212*TE*(614)555-1212*ADDL
CONTACT The figure below shows an example of the same transaction, an
X12 997 Functional Acknowledgment, using standard ANSI format.
```

2.6 Errors and Exceptions

For all HIPAA OTDs, when the incoming message cannot be parsed (for example, if the OTD cannot find the ISA segment), then the *unmarshal()* method generates a com.stc.otd.runtime.UnmarshalException.

You can also use the *isUnmarshalComplete()* method on an OTD to verify whether the *unmarshal()* method completed successfully. Successful completion does not guarantee that the OTD instance is free of data errors within segments and composites because elements are not unmarshaled until the first invocation of the leaf element *getElementXxxx()* method of a segment or composite. For more information, refer to **"On Demand Parsing" on page 12**. Encountering this triggers an automatic background unmarshal of the entire segment. Note that the value returned by the *isUnmarshalComplete()* method is not influenced by the outcome of the automatic background unmarshal; instead, its value reflects what was set by the explicit invocation of the *unmarshal()* method.

Chapter 3

Installing the HIPAA OTDs

This chapter describes how to install HIPAA OTDs, the SEF wizard, and the HIPAA OTD Library documentation.

What's in This Chapter

- **System Requirements** on page 15
- **Supported Operating Systems** on page 15
- **Installing the HIPAA OTD Library** on page 16
- **Increasing the Enterprise Designer Heap Size** on page 17

3.1 System Requirements

Each HIPAA OTD **.sar** file requires approximately 7.5 MB.Tthe combined disk space required to load the standard and Addenda **.sar** files is approximately 15 MB.

Due to the size of the HIPAA OTDs, it is recommended that you increase the heap size property of the Enterprise Designer. For information, refer to **"Increasing the Enterprise Designer Heap Size" on page 17**.

Other than that, the system requirements for the HIPAA OTD Library are the same as those for eGate Integrator and eInsight Business Process Manager. For information, refer to the *SeeBeyond ICAN Suite Installation Guide*.

3.2 Supported Operating Systems

The HIPAA OTD Library is available for the following operating systems:

- Windows XP, Windows 2000, and Windows Server 2003
- HP Tru64 V5.1A
- HP-UX 11.0, 11i (PA-RISC), and 11i v2.0 (11.23)
- IBM AIX 5.1L and 5.2
- Red Hat Enterprise Linux Advanced Server 2.1 (Intel x86)
- Red Hat Linux 8 (Intel x86)

- Sun Solaris 8 and 9

3.3 Installing the HIPAA OTD Library

During the HIPAA OTD Library installation process, the Enterprise Manager, a Web-based application, is used to select and upload products as **.sar** files from the ICAN Suite installation CD-ROM to the Repository.

The installation process includes the following steps:

- Installing the Repository
- Uploading products to the Repository
- Downloading components (such as Enterprise Designer and Logical Host)
- Viewing product information home pages

Follow the instructions for installing the eGate Integrator in the *SeeBeyond ICAN Suite Installation Guide*, and include the steps below to install the HIPAA OTDs. You must have uploaded a **license.sar** to the ICAN Repository that includes a license for the HIPAA OTD Library.

To install the HIPAA OTD Library

1. After uploading the **eGate.sar** or **eInsightESB.sar** file to the ICAN Repository, select and upload the items below as described in the *SeeBeyond ICAN Suite Installation Guide*:
   - **HIPAA_2000_Standard_OTD_Lib.sar** for the standard 2000 HIPAA OTDs
   - **HIPAA_2000_Addenda_OTD_Lib.sar** for the 2000 HIPAA Addenda OTDs
   - **HIPAA_OTD_Docs.sar** (to install the user's guide)
   - **SEF_OTD_Wizard.sar** (to install the SEF OTD wizard from Products CD 3 to be able to build SEF OTDs)
2. Click the **DOCUMENTATION** page, click **HIPAA OTD Library** in the left pane, and click **HIPAA OTD Library User's Guide** to download the documentation in PDF form.
3. Start (or restart) the Enterprise Designer, and click **Update Center** on the **Tools** menu. The Update Center shows a list of components ready for updating.
4. Click **Add All** (the button with a doubled chevron pointing to the right). All modules move from the **Available/New** pane to the **Include in Install** pane.
5. Click **Next** and, in the next window, click **Accept** to accept the license agreement.
6. When the progress bars indicate the download has ended, click **Next**.
7. Review the certificates and installed modules, and then click **Finish**.
8. When prompted to restart Enterprise Designer, click **OK**.

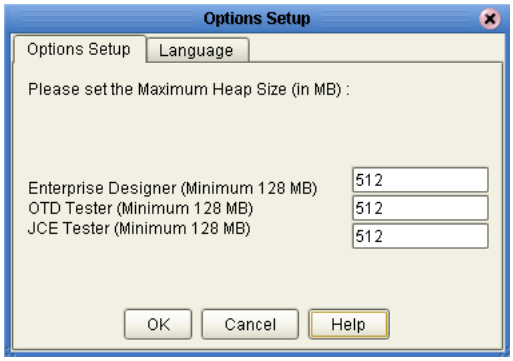
3.4 Increasing the Enterprise Designer Heap Size

Due to the size of the HIPAA OTDs, you may need to increase the heap size property of the Enterprise Designer. If the heap size is not increased, out of memory errors may occur.

To increase the Enterprise Designer heap size

1. On the **Tools** menu in Enterprise Designer, click **Options**. The **Options Setup** dialog box appears.
2. Set the configured heap size for the Enterprise Designer, OTD Tester, and JCE Tester to no less than 512 MB, and click **OK**.

Figure 1 Increasing Enterprise Designer Heap Size



3. Restart Enterprise Designer.

3.4.1 Resolving Memory Errors at Enterprise Designer Startup

If an out of memory error occurs at Enterprise Designer startup, change the setting in the **heapSize.bat** file. This file is resides in the folder ***ICAN_Suite*\edesigner\bin**, where *ICAN_Suite* is the folder where eGate Integrator is installed.

Open the file with a text editor, and change the heap size settings to no less than 512 MB. Save the file, and restart the Enterprise Designer.

Chapter 4

Using HIPAA OTDs

This chapter describes how you use HIPAA OTDs provided in the HIPAA OTD Library, such as customizing OTDs and building HIPAA Collaborations.

What's in This Chapter

- **Displaying HIPAA OTDs** on page 18
- **Building HIPAA OTD Collaborations** on page 19
- **Customizing the HIPAA OTDs** on page 21
- **Creating HIPAA OTDs from SEF Files** on page 23
- **Possible Differences in Output When Using Pass-Through** on page 25

4.1 Displaying HIPAA OTDs

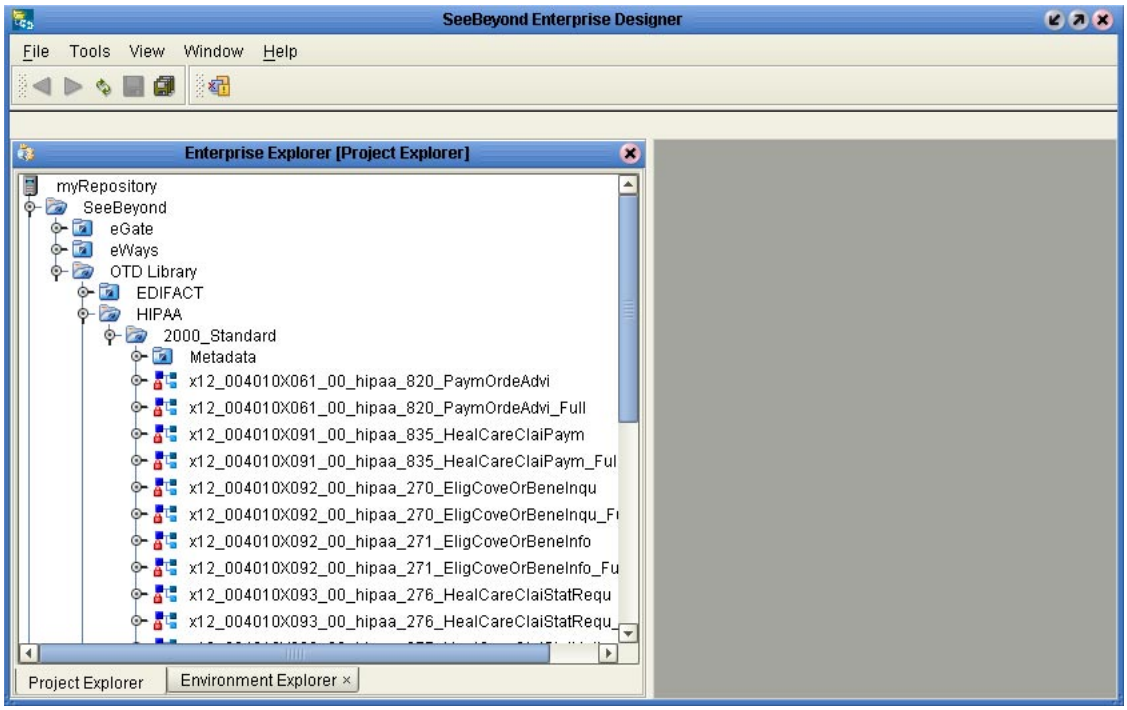
After installing the HIPAA OTDs, you can view the OTDs in the OTD Editor as described below.

To display HIPAA OTDs

1. In the **Project Explorer** tab of Enterprise Designer, expand the following folders:
   - **SeeBeyond**
   - **OTD Library**
   - **HIPAA**
   - **2000_Standard** or **2000_Addenda**

   The Project Explorer tab displays the available OTDs.

Figure 2 OTDs for the 2000 Standard



The table below described the OTD naming conventions.

Table 2 OTD Naming Convention

x12_	X12 protocol name
v*nnnn*_	X12 version
00_	2000 HIPAA
hipaa_	Protocol name
q*n*_	Optional; indicating quarter
820_PayMOrdeAdvi	Transaction code and transaction name abbreviation
_Full	Fully enveloped OTD version that includes the inner and outer envelopes

The folder also includes a **Metadata** folder, which holds a .**zip** file that contains all HIPAA SEF files for the OTDs. You can use the SEF file to customize the OTD as described in **Customizing the HIPAA OTDs** on page 21.

4.2 Building HIPAA OTD Collaborations

This section describes how you build Java Collaborations that use the HIPAA OTDs provided in the HIPAA OTD Library.

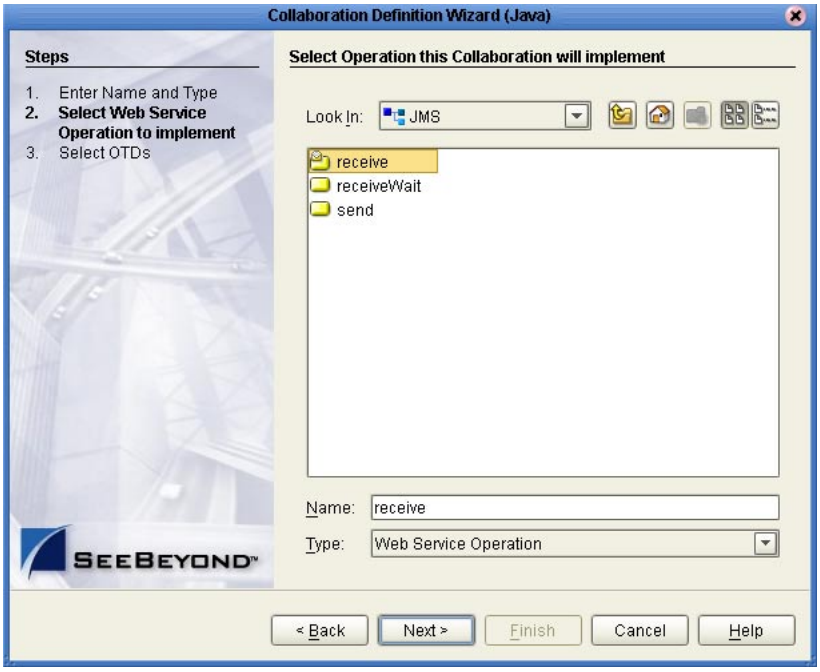
To customize the OTDs before building the Collaboration, refer to **"Customizing the HIPAA OTDs" on page 21**.

Before you can build the Collaboration, you must have installed the **.sar** file for the particular OTD to be used. For information, see **"Installing the HIPAA OTD Library" on page 16**.

To build HIPAA OTD Collaborations

1. In the **Project Explorer** tab of Enterprise Designer, right-click the Project for which you want to create a Collaboration, click **New**, and click **Collaboration Definition (Java)**. The **Collaboration Definition Wizard** dialog box appears.
2. Enter the name of the Collaboration and click **Next**. The **Select Web Service Operation** page appears.
3. Select to the Web service to be used for this Collaboration, for example, **SeeBeyond**>**eGate**>**JMS**>**receive**, and click **Next**.

Figure 3 Selecting the Web Service



The **Select OTDs** page appears.

4. Under **Look In,** navigate to the OTDs by double-click the following folders:
   - **SeeBeyond**
   - **OTD Library**
   - **HIPAA**
   - **2000_Standard** or **2000_Addenda**

The **Look In** area displays the OTDs for the selected HIPAA directories. The table below describes the naming convention for the OTDs.

Table 3 OTD Naming Convention

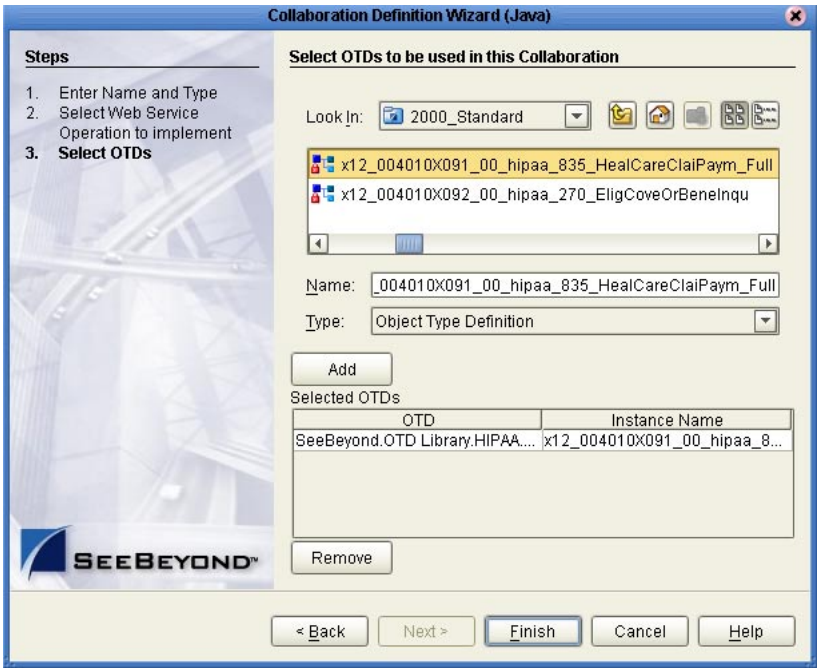
x12_	X12 protocol

Table 3 OTD Naming Convention

v*nnnn*_	X12 version
00_	2000 HIPAA
hipaa_	Protocol name
q1_	Optional; indicating quarter
820_PayMOrdeAdvi	Transaction code and transaction name abbreviation
_Full	Fully enveloped OTD version that includes the inner and outer envelopes

5 Double-click the OTDs to be used. This adds the OTDs under **Selected OTDs**.

Figure 4 Adding OTDs to the Collaboration



6 Click **Finish**. The Collaboration appears in the Collaboration Editor. You can now use the eGate and OTD methods to build the business logic for the Collaboration. For information about the HIPAA OTD methods, refer to **"Java Methods for HIPAA OTDs" on page 26**.

4.3 Customizing the HIPAA OTDs

OTDs provided in the OTD Library cannot be customized. However, the OTD Library provides the SEF file to allow you to modify the file and then rebuild it. The SEF file contains definitions for all transaction sets of the HIPAA X12 version.

You can then rebuild the OTD with the customized SEF file as described in the following section. The procedure below describes how to save the SEF files locally for editing.

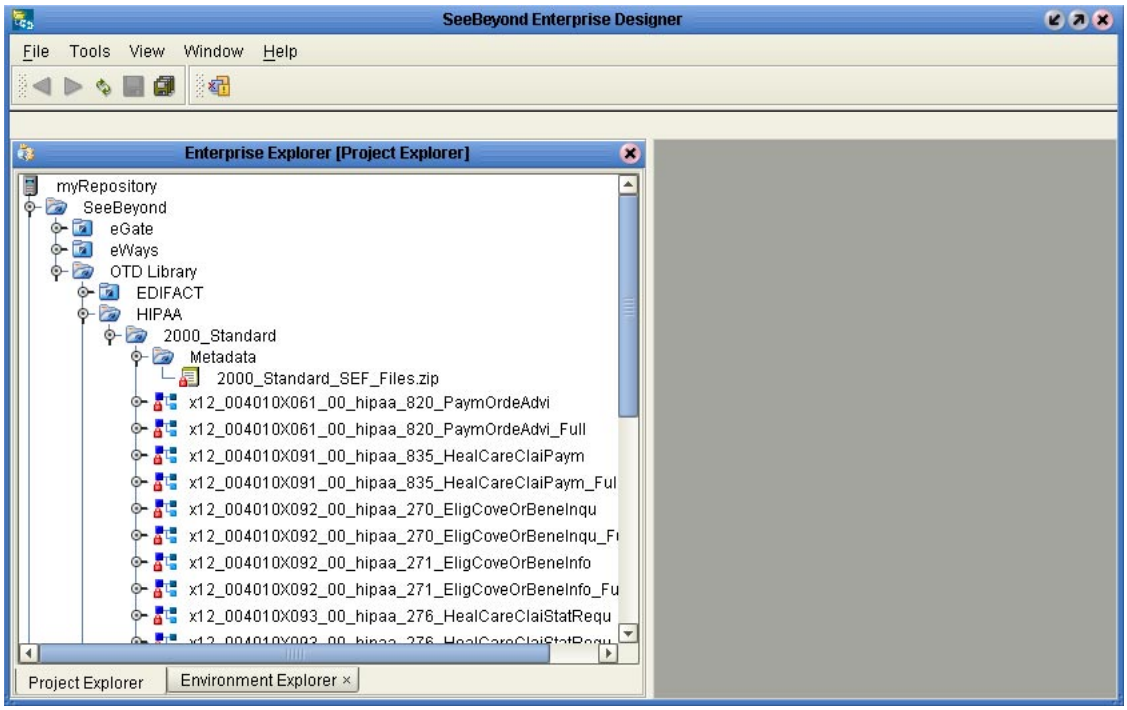
To customize HIPAA OTDs

1 In the **Project Explorer** tab of Enterprise Designer, expand the following folders:

- **SeeBeyond**
- **OTD Library**
- **HIPAA**
- **2000 Standard** or **2000 Addenda**
- **Metadata**

The metadata folder displays a **.zip** file containing all HIPAA SEF files.

Figure 5 Saving HIPAA OTD SEF Files



2 Right-click the **.zip** file to be customized and click **Export**. The **Save As** dialog box appears.

3 Select a location for the **.zip** file and click **Save**.

4 Extract the **.zip** file.

5 Use a SEF editor to customize the extracted file(s).

6 Use the SEF OTD wizard to rebuild the OTD as described in the next section.

4.4 Creating HIPAA OTDs from SEF Files

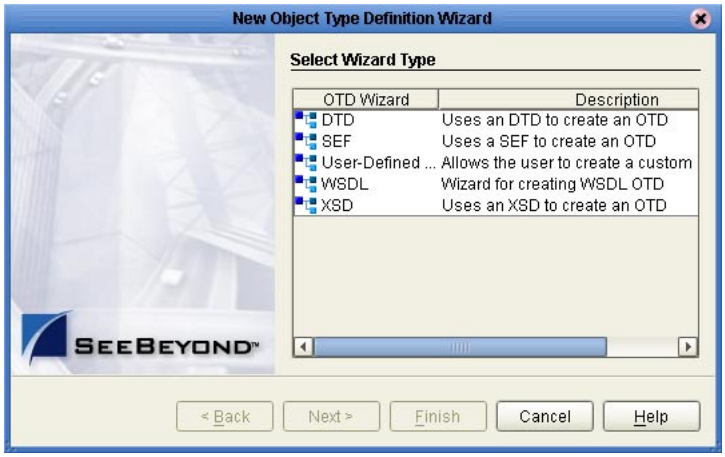
This section describes how you create HIPAA OTDs using SEF files. The HIPAA OTD Library includes the SEF files for the OTDs to allow you to customize the OTD as described in the section above. Once you have tailored the SEF file to your business requirements, you can then use the procedure below to recreate the OTD.

To create OTDs from SEF files, you use the SEF OTD wizard to build the OTD using selected SEF files. The SEF OTD wizard is packaged separately from the OTD Library, so make sure that you uploaded the **SEF_OTD_Wizard.sar** to the ICAN Repository, and used the **Update Center** in Enterprise Designer to install it. For information, refer to **"Installing the HIPAA OTD Library" on page 16**.

To create HIPAA OTDs from SEF files

1 In the Explorer tab of the Enterprise Designer, right click the Project, click **New**, and click **Object Type Definition**. The **New Object Type Definition** dialog box appears.

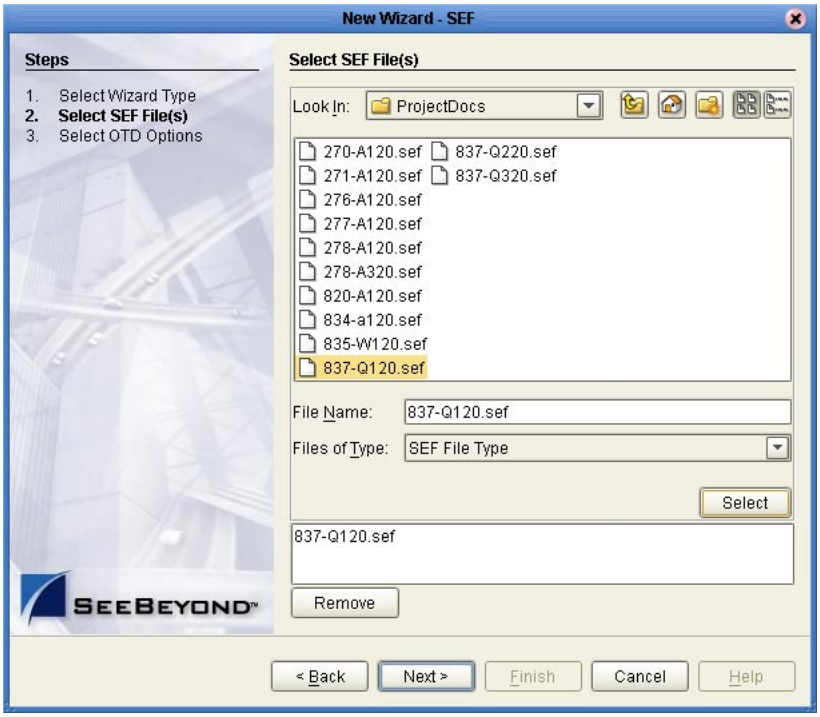
Figure 6 Creating HIPAA OTDs



2 Click **SEF** and click **Next**. The **Select SEF File(s)** page appears.

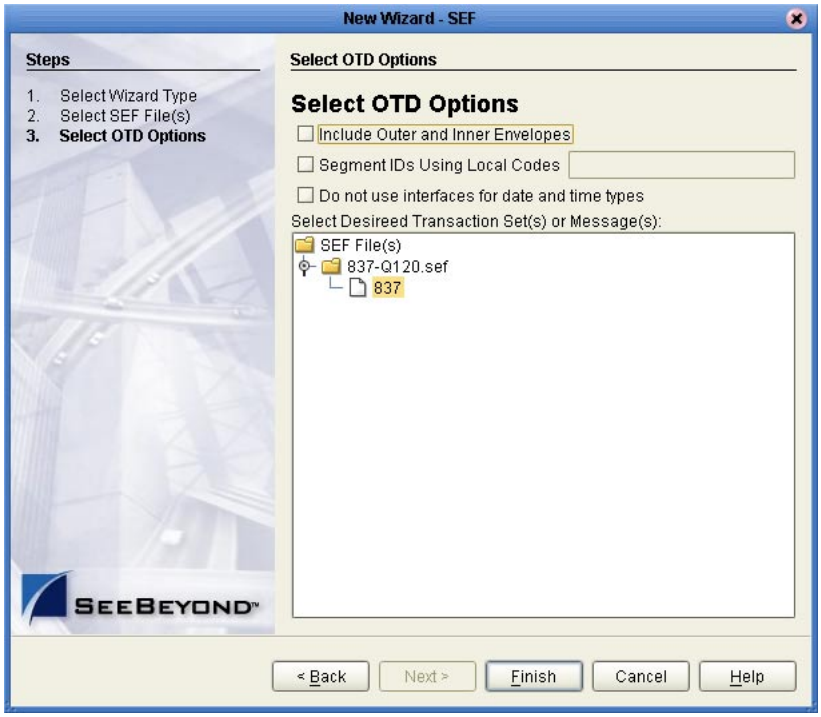
3 In the **Look In** box, navigate to the folder where the SEF file for this OTD resides, and then double-click the SEF file. This adds the file to the selection box as shown below.

Figure 7 Selecting the SEF File



4 Click **Next**. The **Select OTD Options** page appears.

Figure 8 Selecting the OTD Options



5 To include the inner and outer envelopes, select the **Include Outer and Inner Envelopes** option.

6 To use local codes for segment IDs, select the **Segment IDs Using Local Codes** option and enter the code.

7 To avoid the OTD using eInsight interfaces for date and time types, select the **Do Not Use Interfaces for Date and Time Types** option.

Select this option to make the OTD compatible with Collaborations that were created with earlier X12 OTD Library versions.

Selecting this option creates OTDs that do not support mapping a date or time node to another node of eInsight's date and time type in a Business Process.

8 Click **Finish**. The OTD Editor appears, displaying the OTD.

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.

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, 040715 in the input file might be represented as 20040705 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

Java Methods for HIPAA OTDs

This chapter describes the Java methods available for HIPAA OTDs.

What's in This Chapter

- **Get and Set Methods** on page 26
- **Setting Delimiters** on page 27
- **Available Methods** on page 28

5.1 Get and Set Methods

The OTDs in the HIPAA OTD Library contain the Java methods that enable you to set and get the delimiters, which in turn extend the functionality of the HIPAA OTD Library.

The following get and set methods are available under the root node and at the *xxx*_Outer, *xxx*_Inner, and *xxx* levels:

- **setDefaultX12Delimiters** on page 37
- **getElementSeparator** on page 30 and **setElementSeparator** on page 38
- **getFGValidationResult** on page 30
- **getICValidationResult** on page 31
- **getInputSource** on page 31
- **getMaxDataError** on page 32 and **setMaxDataError** on page 38
- **getMaxFreedSegsComsNum** on page 32 and **setMaxFreedSegsComsNum** on page 39
- **getMaxParsedSegsComsNum** on page 32and **setMaxParsedSegsComsNum** on page 39
- **getMsgValidationResult** on page 32
- **getRepetitionSeparator** on page 33 and **setRepetitionSeparator** on page 39
- **getSegmentCount** on page 33
- **getSegmentTerminator** on page 33 and **setSegmentTerminator** on page 39
- **getSubelementSeparator** on page 34 and **setSubelementSeparator** on page 40

- **getTSValidationResult** on page 34
- **getUnmarshalError** on page 34
- **getXmlOutput** on page 34

The following methods are available from the loop elements:

- **getLoopxxx** on page 31 and **setLoopxxx** on page 38
- **getSegmentCount** on page 33
- **setXmlOutput** on page 40

Note: *The get and set methods are automatically generated from the bean nodes. On occasion, this means get and set methods may be available that are not beneficial, such as setFGValidationResult.*

5.2 Setting Delimiters

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

- Data element separator (default is an asterisk)
- Subelement separator/component element separator (default is a colon)
- Repetition separator (default is a plus sign)
- Segment terminator (default is a tilde)

The repetition separator and subelement separator are explicitly specified in the interchange header segment. The other two delimiters are implicitly defined within the structure of the interchange header segment, by their first use. For example, after the fourth character defines the data element separator, the same character is used subsequently to delimit all data elements; and after the 107th character defines the segment terminator, the same character is used subsequently to delimit all segments.

5.2.1 Incoming Message

Because the fully-enveloped OTD automatically detects delimiters in an incoming message that has the interchange header segment while unmarshaling, do not specify delimiters for the incoming message. Any delimiters that are set before unmarshaling are ignored, and the *unmarshal()* method picks up the delimiter used in the ISA segment of the incoming message.

For non-enveloped OTDs, if the incoming message uses non-standard delimiters, set the delimiters on the OTD instance before the *unmarshal()* method is invoked.

You can set the delimiters in the Java Collaboration Editor using the methods or bean nodes that are provided in the OTDs. Use the following methods to specify delimiters:

- **setDefaultX12Delimiters** on page 37
- **setElementSeparator** on page 38

- **setSegmentTerminator** on page 39
- **setSubelementSeparator** on page 40
- **setRepetitionSeparator** on page 39
- **setSubelementSeparator** on page 40)

If the input data is already in HIPAA format, you can use the get methods to retrieve the delimiters from the input data. For information, refer to **"Get and Set Methods" on page 26**.

5.2.2 Outgoing Message

If an OTD outputs ANSI X12 data rather than XML, you must specify the delimiters only if non-standard delimiters are used. If the delimiters are not specified, the industry standard delimiters are used. For information about which methods to use for delimiter setting, refer to the section above.

For fully-enveloped OTDs, you can also set the subelement separator and repetition separator from the corresponding elements within the ISA segment.

5.3 Available Methods

This section describes the signature and description for each available HIPAA OTD method.

check

Signature

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

Description

Validates the content of the OTD data tree at runtime and returns a string array of validation errors for the message body only; validation errors for envelope segments are not included. To include envelope segments, see the checkAll() method below.

The method returns null if there are no validation errors.

Exceptions

None.

checkAll

Signature

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

Description

Validates the content of the OTD data tree at runtime and returns a string array of validation errors for the message body and the envelope segments. The checkAll() method is only available for fully-enveloped OTDs.

The method returns null if there are no validation errors.

Exceptions

None.

---

clone

Signature

```
public java.lang.Object clone()
```

Description

Creates and returns a copy of this OTD instance.

Exceptions

java.lang.CloneNotSupportedException

---

count*xxx*

Signature

```
public int countxxx()
```

where *xxx* is the bean name for repeatable nodes.

Description

Counts the repetitions of the node at runtime.

Exceptions

None.

---

countLoop*xxx*

Signature

```
public int countLoopxxx()
```

where *xxx* is the bean node for a repeatable segment loop.

Description

Counts the repetitions of the loop at runtime.

Exceptions

None.

getxxx

Signature

```
public item getxxx()
```

where *xxx* is the bean name for the node and where *item* is the Java type for the node.

```
public item[] getxxx()
```

where *xxx* is the bean name for the repeatable node and where *item[]* is the Java type for the node.

Description

Returns the node object or the object array for the node.

Exceptions

None.

getAllErrors

Signature

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

Description

Returns all the validation errors as a string array. These validation errors include errors encountered during unmarshaling input data and the validation results from both the message and the envelope segments.

Exceptions

None.

getElementSeparator

Signature

```
public char getElementSeparator()
```

Description

Gets the element separator character.

Exceptions

None.

getFGValidationResult

Signature

```
public com.stc.otd.runtime.edi.FGError[] getFGValidationResult()
```

Description

Returns the validation errors for the functional group envelope in the format of an FGError array. This method is available only at the Outer and Inner root levels in fully-enveloped OTDs.

Exceptions

None.

getICValidationResult

Signature

```
public com.stc.otd.runtime.edi.ICError[] getICValidationResult()
```

Description

Returns the validation errors for the interchange envelope in the format of an ICError array. This method is available only at the Outer and Inner root levels in fully-enveloped OTDs.

Exceptions

None.

getInputSource

Signature

```
public byte[] getInputSource()
```

Description

Returns the byte array of the original input data source.

Exceptions

None.

getLoopxxx

Signature

```
public item getLoopxxx()
```

where *Loopxxx* is the bean name for the segment loop and where *item* is the Java type for the segment loop.

```
public item[] getLoopxxx()
```

where *Loopxxx* is the bean name for the repeatable segment loop and where *item[]* is the Java type for the repeatable segment loop.

Description

Returns the segment loop object or the object array for the segment loop.

Exceptions

None.

getMaxDataError

Signature

```
public int getMaxDataError()
```

Description

Returns the maximum number of message validation errors held in the *msgValidationResult* bean node. If this method returns -1 there is no limit of how many errors can be reported.

Exceptions

None.

getMaxFreedSegsComsNum

Signature

```
public int getMaxFreedSegsComsNum()
```

Description

Returns the maximum number of segment and composite objects marked to be freed from memory. For more information, refer to **"On Demand Parsing" on page 12**.

Exceptions

None.

getMaxParsedSegsComsNum

Signature

```
public int getMaxParsedSegsComsNum()
```

Description

Returns the maximum number of segments and composite objects to be parsed. For more information, refer to **"On Demand Parsing" on page 12**.

Exceptions

None.

getMsgValidationResult

Signature

```
public com.stc.otd.runtime.check.sef.DataError[]
getMsgValidationResult()
```

Description

Returns the validation errors for the message body. Use this method after the *performValidation()* method. For information, refer to **"performValidation" on page 36**.

This method is only available at the Outer, Inner, and transaction set levels. It is also available at the top root level of non-enveloped OTDs.

Exceptions

None.

getRepetitionSeparator

Signature

```
public char getRepetitionSeparator()
```

Description

Returns the repetition separator character.

Exceptions

None.

getSegmentCount

Signature

```
public int getSegmentCount()
```

Description

Returns the segment count at the current level.

Exceptions

None.

getSegmentTerminator

Signature

```
public char getSegmentTerminator()
```

Description

Returns the segment terminator character.

Exceptions

None.

getSubelementSeparator

Signature

```
public char getSubelementSeparator()
```

Description

Returns the subelement/composite element separator character.

Exceptions

None.

getTSValidationResult

Signature

```
public com.stc.otd.runtime.edi.TSError[] getTSValidationResult()
```

Description

Returns the validation errors for the message envelope (segments UNH/UIH and UNT/UIT) in the format of an TSError array. This method is available only at the Outer, Inner, and transaction set root levels of fully enveloped OTDs. It is also available at the root level of non-enveloped OTDs.

Exceptions

None.

getUnmarshalError

Signature

```
public com.stc.otd.runtime.check.sef.DataError[] getUnmarshalError()
```

Description

Returns the unmarshal errors as an array of the DataError objects. The unmarshal errors are reported from an UnmarshalException generated during unmarshaling. Usually these errors are associated with otd.isUnmarshalComplete=false.

Exceptions

None.

getXmlOutput

Signature

```
public boolean getXmlOutput()
```

Description

Verifies whether the X12 OTD will output data in XML format. For more information, refer to **"Alternative Formats: ANSI and XML" on page 12**.

Exceptions

None.

Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
boolean isXml=myOTD.getXmlOutput();
```

hasxxx

Signature

```
public boolean hasxxx()
```

where *xxx* is the bean name for the node.

Description

Verifies if the node is present in the runtime data.

Exceptions

None.

has*Loopxxx*

Signature

```
public boolean hasLoopxxx()
```

where *Loopxxx* is the bean name for the segment loop.

Description

Verifies if the segment loop is present in the runtime data.

Exceptions

None.

isUnmarshalComplete

Signature

```
public boolean isUnmarshalComplete()
```

Description

Flag for whether or not unmarshaling completed successfully. For more information, see **"On Demand Parsing" on page 12** and **"Errors and Exceptions" on page 14**.

Exceptions

None.

marshal

Signature

```
public void marshal(com.stc.otd.runtime.OtdOutputStream)
```

Description

Marshals the internal data tree into an output stream.

Exceptions

java.io.IOException for output problems

com.stc.otd.runtime.MarshalException for an inconsistent internal tree

marshalToBytes

Signature

```
public byte[] marshalToBytes()
```

Description

Marshals the internal data tree into a byte array.

Exceptions

java.io.IOException for output problems

com.stc.otd.runtime.MarshalException for an inconsistent internal tree

marshalToString

Signature

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

Description

Marshals the internal data tree into a String.

Throws

java.io.IOException for input problems

com.stc.otd.runtime.MarshalException for an inconsistent internal tree

performValidation

Signature

```
public void performValidation()
```

Description

Performs validation on the OTD instance unmarshaled from input data.

You can access the validation results from a list of nodes, such as allErrors, msgValidationResult, and the node for reporting envelope errors (such as ICValidationResult, FGValidationResult, and TSValidationResult). For more information, refer to **"HIPAA Validation Support" on page 11**.

Exceptions

None.

---

reset

Signature

```
public void reset()
```

Description

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

Exceptions

None.

---

set*xxx*

Signature

```
public void setxxx(item)
```

where *xxx* is the bean name for the node and where *item* is the Java type for the node.

```
public void setxxx(item[])
```

where *xxx* is the bean name for the repeatable node and where *item[]* is the Java type for the node.

Description

Sets the node object or the object array for the node.

Exceptions

None.

---

setDefaultX12Delimiters

Signature

```
public void setDefaultX12Delimiters()
```

Description

Sets the current delimiters to the default HIPAA delimiters:

- segment terminator = ~
- element separator = *
- subelement separator = :

- repetition separator = +

For more information ,refer to **"Setting Delimiters" on page 27**.

Exceptions

None.

---

setElementSeparator

Signature

```
public void setElementSeparator(char arg0)
```

Description

Sets the element separator character. For more information ,refer to **"Setting Delimiters" on page 27**.

Exceptions

None

---

setLoopxxx

Signature

```
public void setLoopxxx(item)
```

where *Loopxxx* is the bean name for the segment loop and where *item* is the Java type for the segment loop.

```
public void setLoopxxx(item[])
```

where *Loopxxx* is the bean name for the repeatable segment loop and where *item[]* is the Java type for the repeatable segment loop.

Description

Sets the segment loop object or the object array for the segment loop.

Exceptions

None.

---

setMaxDataError

Signature

```
public void setMaxDataError(int)
```

Description

Returns the maximum number of message validation errors held in the *msgValidationResult* bean node. If this method returns -1 there is no limit of how many errors can be reported.

Exceptions

None.

setMaxFreedSegsComsNum

Signature

```
public void setMaxFreedSegsComsNum(int)
```

Description

Sets the maximum number of segment and composite objects marked to be freed from memory. For more information, refer to **"On Demand Parsing" on page 12**.

Exceptions

None.

setMaxParsedSegsComsNum

Signature

```
public void setMaxParsedSegsComsNum(int)
```

Description

Sets the maximum number of segments and composite objects to be parsed. For more information, refer to **"On Demand Parsing" on page 12**.

Exceptions

None.

setRepetitionSeparator

Signature

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

Description

Sets the repetition separator character. For more information, refer to **"Setting Delimiters" on page 27**.

Exceptions

None.

setSegmentTerminator

Signature

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

Description

Sets the segment terminator character. For more information, refer to **"Setting Delimiters" on page 27**.

Exceptions

None.

setSubelementSeparator

Signature

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

Description

Sets the subelement separator character. For more information, refer to **"Setting Delimiters" on page 27**.

Exceptions

None.

setXmlOutput

Signature

```
public void setXmlOutput(boolean)
```

Description

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

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

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

For more information, refer to **"Alternative Formats: ANSI and XML" on page 12**.

Exceptions

None.

unmarshal

Signature

```
public void unmarshal(com.stc.otd.runtime.OtdInputStream)
```

Description

Unmarshals the given input into an internal data tree.

For more information, refer to in **"On Demand Parsing" on page 12** and **"Errors and Exceptions" on page 14**.

Exceptions

java.io.IOException for output problems

com.stc.otd.runtime.UnmarshalException for a lexical or other mismatch

unmarshalFromBytes

Signature

```
public void unmarshalFromBytes(byte[])
```

Description

Unmarshals the given input byte array into an internal data tree.

Exceptions

java.io.IOException for input problems

com.stc.otd.runtime.UnmarshalException for an inconsistent internal tree

unmarshalFromString

Signature

```
public void unmarshalFromString(java.lang.String)
```

Description

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

Exceptions

java.io.IOException for input problems

com.stc.otd.runtime.UnmarshalException for an inconsistent internal tree. This typically occurs when the OTD does not recognize the incoming message as X12.

Appendix A

X12OTDErrors Schema File and Sample XML

This appendix provides the contents of the X12OTDErrors.xsd file, which is the schema file the validation output string conforms to. This appendix also provides a sample of validation output XML.

For more information, refer to **"HIPAA Validation Support" on page 11** and **"performValidation" on page 36**.

What's in This Chapter

- **Contents of the X12OTDErrors.xsd File** on page 42
- **Sample of Validation Output XML** on page 43

6.1 Contents of the X12OTDErrors.xsd File

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by Tony (TechLeader) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="X12OTDErrors">
    <xs:annotation>
      <xs:documentation>Validation Errors from an HIPAA OTD validation</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="X12ICError" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="X12FGError" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="X12TSError" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="X12DataError" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="X12ICError">
    <xs:annotation>
      <xs:documentation>Interchange Envelope Validation Error Structure. For TA1 generations</
xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="InteContNumb" type="xs:string"/>
        <xs:element name="InteContDate" type="xs:string"/>
        <xs:element name="InteContTime" type="xs:string"/>
        <xs:element name="InteNoteCode" type="xs:string"/>
        <xs:element name="ICErrorDesc" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="X12FGError">
    <xs:annotation>
      <xs:documentation>Functional Group Envelope Validation Error Structure. For AK1AK9
generations</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="FuncIdenCode" type="xs:string"/>
```

```
        <xs:element name="GrouContNumb" type="xs:string"/>
        <xs:element name="NumbOfTranSetsIncl" type="xs:string"/>
        <xs:element name="FuncGrouSyntErroCode" type="xs:string"/>
        <xs:element name="FGErrorDesc" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="X12TSError">
    <xs:annotation>
      <xs:documentation>Transaction Set Envelope Validation Error Structure. For AK2AK5
generations</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="TranSetIdenCode" type="xs:string"/>
        <xs:element name="TranSetContNumb" type="xs:string"/>
        <xs:element name="TranSetSyntErroCode" type="xs:string"/>
        <xs:element name="TSErrorDesc" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="X12DataError">
    <xs:annotation>
     <xs:documentation>Transaction Set (excluding envelopes) Validation Error Structure. For AK3AK4
generations</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Level" type="xs:short" minOccurs="0"/>
        <xs:element name="SegmIDCode" type="xs:string"/>
        <xs:element name="SegmPosiInTranSet" type="xs:int"/>
        <xs:element name="LoopIdenCode" type="xs:string" minOccurs="0"/>
        <xs:element name="SegmSyntErroCode" type="xs:short" minOccurs="0"/>
        <xs:element name="ElemPosiInSegm" type="xs:short"/>
        <xs:element name="CompDataElemPosiInComp" type="xs:short" minOccurs="0"/>
        <xs:element name="DataElemRefeNumb" type="xs:string" minOccurs="0"/>
        <xs:element name="DataElemSyntErroCode" type="xs:short"/>
        <xs:element name="CopyOfBadDataElem" type="xs:string" minOccurs="0"/>
        <xs:element name="RepeatIndex" type="xs:short" minOccurs="0"/>
        <xs:element name="ErrorCode" type="xs:int"/>
        <xs:element name="ErrorDesc" type="xs:string" minOccurs="0"/>
        <xs:element name="Severity" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

6.2 Sample of Validation Output XML

```
<X12OTDErrors>
  <X12ICError>
    <InteContNumb>000000001</InteContNumb>
    <InteContDate>041102</InteContDate>
    <InteContTime>1441</InteContTime>
    <InteNoteCode>021</InteNoteCode>
    <ICErrorDesc>Invalid Number of Included Groups Value</ICErrorDesc>
  </X12ICError>
  <X12FGError>
    <FuncIdenCode>PO</FuncIdenCode>
    <GrouContNumb>1</GrouContNumb>
    <NumbOfTranSetsIncl>2</NumbOfTranSetsIncl>
    <FuncGrouSyntErroCode>5</FuncGrouSyntErroCode>
  </X12FGError>
  <X12FGError>
    <FuncIdenCode>PO</FuncIdenCode>
    <GrouContNumb>1</GrouContNumb>
    <NumbOfTranSetsIncl>2</NumbOfTranSetsIncl>
    <FuncGrouSyntErroCode>4</FuncGrouSyntErroCode>
    <FGErrorDesc>Number of Included Transaction Sets Does Not Match Actual Count</FGErrorDesc>
  </X12FGError>
  <X12TSError>
    <TranSetIdenCode>850</TranSetIdenCode>
    <TranSetContNumb>0001</TranSetContNumb>
    <TranSetSyntErroCode>4</TranSetSyntErroCode>
    <TSErrorDesc>Number of Included Segments Does Not Match Actual Count</TSErrorDesc>
  </X12TSError>
  <X12DataError>
    <Level>1</Level>
    <SegmIDCode>MEA</SegmIDCode>
    <SegmPosiInTranSet>21</SegmPosiInTranSet>
    <LoopIdenCode/>
    <SegmSyntErroCode>8</SegmSyntErroCode>
    <ElemPosiInSegm>4</ElemPosiInSegm>
    <DataElemRefeNumb>C001</DataElemRefeNumb>
    <DataElemSyntErroCode>10</DataElemSyntErroCode>
    <ErrorCode>15025</ErrorCode>
```

```
    <ErrorDesc>MEA_4 at 21: [Syntax rule E-Exclusion: One or None] Exclusion condition violated
because E0412</ErrorDesc>
    <Severity>ERROR</Severity>
  </X12DataError>
  <X12DataError>
    <Level>1</Level>
    <SegmIDCode>N4</SegmIDCode>
    <SegmPosiInTranSet>195</SegmPosiInTranSet>
    <LoopIdenCode>N1</LoopIdenCode>
    <SegmSyntErroCode>8</SegmSyntErroCode>
    <ElemPosiInSegm>7</ElemPosiInSegm>
    <DataElemRefeNumb>1715</DataElemRefeNumb>
    <DataElemSyntErroCode>10</DataElemSyntErroCode>
    <CopyOfBadDataElem>CNT</CopyOfBadDataElem>
    <ErrorCode>15025</ErrorCode>
    <ErrorDesc>N1_N4_7 at 195 [CNT]: [Syntax rule E-Exclusion: One or None] Exclusion condition
violated because E0207</ErrorDesc>
    <Severity>ERROR</Severity>
  </X12DataError>
</X12OTDErrors>
```

Index

A

AllErrors **30**

C

check() method **28**
checkAll() method **28**
clone() method **29**
Collaborations, building **19**
component element separator **27**
conventions, document **8**
count() method **29**
countLoopxxx() method **29**
customizing OTDs **21**

D

data element separator **27**
delimiters **11**, **27**
  component element separator **27**
  data element separator **27**
  repetition separator **27**
  segment terminator **27**
  subelement separator **27**
displaying OTDs **18**
document conventions **8**

E

EDFOTDErrors.xsd **42**
elementSeparator **30**, **38**
Exceptions
  IOException **36**, **41**
  MarshalException **36**
  UnmarshalException **41**

F

FGError **30**
FGValidationResult **30**

G

get methods, overview **26**
getAllErrors() method **30**
getElementSeparator() method **30**
getFGValidationResult() method **30**
getICValidationResult() method **31**
getInputSource() method **31**
getLoopxxx() method **31**
getMaxDataError() method **32**
getMaxFreedSegsComsNum() method **32**
getMaxParsedSegsComsNum() method **32**
getMsgValidationResult() method **32**
getRepetitionSeparator() method **33**
getSegmentCount() method **33**
getSegmentTerminator() method **33**
getSubelementSeparator() method **34**
getTSValidationResult() method **34**
getUnmarshalError() method **34**
getXmlOutput() method **34**
getxxx() method **30**

H

hasLoopxxx() method **35**
hasxxx() method **35**
heap size, adjusting **17**

I

ICError **31**
ICValidationResult **31**
inputSource **31**
isUnmarshalComplete() method **35**

M

marshal() method **36**
marshaling
  marshal() **36**
  marshalToBytes() **36**
  marshalToString() **36**
marshalToBytes() method **36**
marshalToString() method **36**
maxDataError **38**
maxFreedSegsComsNum **39**
maxParsedSegsComsNum **32**, **39**
memory
  management **12**
memory errors, resolving **17**
message structure
  defined **11**
  OTD in eGate **11**
methods
  check **28**
  checkAll **28**

clone() **29**
count() **29**
countLoopxxx() **29**
get/set methods, overview **26**
getAllErrors() **30**
getElementSeparator() **30**
getFGValidationResult() **30**
getICValidationResult() **31**
getInputSource() **31**
getLoopxxx() **31**
getMaxDataError() **32**
getMaxFreedSegsComsNum() **32**
getMaxParsedSegsComsNum() **32**
getMsgValidationResult() **32**
getRepetitionSeparator() **33**
getSegmentCount() **33**
getSegmentTerminator() **33**
getSubelementSeparator() **34**
getTSValidationResult() **34**
getUnmarshalError() **34**
getXmlOutput() **34**
getxxx() **30**
hasLoopxxx() **35**
hasxxx() **35**
isUnmarshalComplete() **35**
marshal() **36**
marshalToBytes() **36**
marshalToString() **36**
performValidation() **36**
reset() **37**
setDefaultX12Delimiters() **37**
setElementSeparator() **38**
setLoopxxx() **38**
setMaxDataError() **38**
setMaxFreedSegsComsNum() **39**
setMaxParsedSegsComsNum() **39**
setRepetitionSeparator() **39**
setSegmentTerminator() **39**
setSubelementSeparator() **40**
setXmlOutput() **40**
setxxx() **37**
unmarshal() **40**
unmarshalFromBytes() **41**
unmarshalFromString() **41**
msgValidationResult **32**

O

on demand parsing **12**
organization of information, document **7**
OTDs
Collaborations, using in **19**
customizing **21**
displaying **18**
performValidation() method **36**
reset() method **37**
SEF file, creating from **23**
SEF files **21**
OutOfMemoryError
increase heap size **17**

P

parse on demand **12**
performValidation() method **36**

R

related documents **8**
repetition separator **27**
repetitionSeparator **33**, **39**
reset() method **37**
runtime exceptions
UnmarshalException **14**

S

Screenshots **8**
SEF file **11**
creating OTD from **23**
OTD, customizing **21**
SEF OTD wizard
installing **16**
using **23**
segment terminator **27**
segmentCount **33**
segmentTerminator **33**, **39**
set methods, overview **26**
setDefaultX12Delimiters() method **37**
setElementSeparator() method **38**
setLoopxxx() method **38**
setMaxDataError() method **38**
setMaxFreedSegsComsNum() method **39**
setMaxParsedSegsComsNum() method **39**
setRepetitionSeparator() method **39**
setSegmentTerminator() method **39**
setSubelementSeparator() method **40**
setXmlOutput() method **40**
setxxx() method **37**
subelement separator **27**
subelementSeparator **34**, **40**
support
SEF file **11**
validation **11**

T

TSvalidationResult **34**

U

unmarshal() method **40**
unmarshalError **34**
UnmarshalException **14**
unmarshalFromBytes() method **41**
unmarshalFromString() method **41**
unmarshaling
  delayed **12**
  isUnmarshalComplete() **35**
  unmarshal() method **40**
  unmarshalFromBytes() method **41**
  unmarshalFromString() method **41**

V

validation
  EDFOTDErrors.xsd **42**
  performValidation() method **36**
  reset() method **37**
  support **11**