SUN B2B SUITE 2.0

ASC X12 OTD LIBRARY USER'S GUIDE

Release 5.1.0

Sun microsystems

Part Number: 820-0267-10

Version 20070213130454

Contents

List of Figures

Chapter 1

Introduction

This guide explains the Sun SeeBeyond ASC X12 OTD Library of the Sun Java™ Composite Application Platform Suite of products.

This chapter also provides an overview of the purpose, scope, use, and organization of the document.

What's in This Chapter

1.1 About the ASC X12 OTD Library

The ASC X12 OTD Library includes transaction set structures for each of the transactions available for a given ASC X12 version. You can use these structures as provided or customize them as needed.

eGate Integrator uses Object Type Definitions (OTDs) based on ASC X12 message structures to verify that the data in messages is in the correct format. There is a message structure for each ASC X12 transaction.

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

For a more detailed overview of the ASC X12 OTD Library and its use, see **Chapter 2**.

1.2 About This Document

This section provides information about this document, such as an overview of its contents, scope, and intended audience.

1.2.1 What's In This Document

This document contains the following information:

- Chapter 1, **"Introduction"**, provides a general preview of this document, its purpose, scope, and organization.
- Chapter 2, **"Overview of ASC X12 OTD Library"**, provides an overview of the ASC X12 OTD Library as well as its support for X12 versions, SEF file versions, and validation.
- Chapter 3, **"Installing the ASC X12 OTD Library"**, describes how to install ASC X12 OTDs, the SEF OTD wizard, and the ASC X12 OTD Library documentation.
- Chapter 4, **"Using ASC X12 OTDs"**, describes how to display and customize OTDs, and how to build Collaborations with ASC X12 OTDs.
- Chapter 5, **"Java Methods for ASC X12 OTDs"**, provides the syntax for the Java methods provided with the ASC X12 OTDs.
- Appendix A, **"X12OTDErrors Schema File and Sample XML"**, provides the X12OTDErrors schema file and a sample validation output Extensible Markup Language (XML).

1.2.2 Scope

This document describes the X12 OTD library and how to install and 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.

1.2.3 Intended Audience

This guide is intended for experienced computer users who have the responsibility of helping to set up and maintain a fully functioning Java Composite Application Platform Suite system. This person must also understand any operating systems on which the Java Composite Application Platform Suite will be installed (Windows and UNIX), and must be thoroughly familiar with Windows-style GUI operations.

1.2.4 Text Conventions

The following conventions are observed throughout this document.

Table 1 Text Conventions

Text Convention	Used For	Examples
Bold	Names of buttons, files, icons, parameters, variables, methods, menus, and objects	▪ Click **OK**. ▪ On the **File** menu, click **Exit**. ▪ Select the **eGate.sar** file.
`Monospaced`	Command line arguments, code samples; variables are shown in ***bold italic***	`java -jar` ***`filename`***`.jar`
Blue bold	Hypertext links within document	See **Text Conventions** on page 8
Blue underlined	Hypertext links for Web addresses (URLs) or email addresses	http://www.sun.com

1.2.5 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.

1.2.6 Related Documents

For information on eXchange Integrator, refer to the *eXchange Integrator User's Guide*.

For information on the ASC X12 Protocol Manager, refer to the *ASC X12 Protocol Manager User's Guide*.

For late-breaking information on the B2B Suite, refer to the B2B_Readme file, located on the product media.

Sun SeeBeyond Java Composite Application Platform Suite (CAPS)

For information aboutJava CAPS products, refer to the following:

Title	Filename
Java CAPS Installation Guide	CAPS_Install_Guide.pdf
Java CAPS Deployment Guide	CAPS_Deployment_Guide.pdf
eGate Integrator User's Guide	eGate_UG.pdf
eGate Integrator System Administration Guide	eGate_Sys_Admin_Guide.pdf
eGate Integrator JMS Reference Guide	eGate_JMS_Ref.pdf
eInsight Business Process Manager User's Guide	eInsight_UG.pdf
Readme for Java CAPS / eGate Integrator	Readme.txt

1.3 References

The following Web sites provide additional information about the ASC X12 protocol:

- http://www.disa.org
- http://www.x12.org/x12org/index.cfm
- http://www.wpc-edi.com

1.4 Sun Microsystems, Inc. Web Site

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

http://www.sun.com

1.5 Documentation Feedback

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

CAPS_docsfeedback@sun.com

Chapter 2

Overview of ASC X12 OTD Library

This chapter provides an overview of the ASC X12 OTD Library as well as its support for ASC X12 directory versions, SEF file versions, and validation.

What's in This Chapter

2.1 ASC X12 OTD Library: Introduction

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. The result was the X12 standard. This standard allows the electronic exchange of machine-readable information between businesses.

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

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

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

X12 implementation guides can be obtained at the following Web sites:

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

ASC X12 messages have a message structure, which indicates how data elements are organized and related to each other for a particular EDI transaction. In the Sun Java Composite Application Platform Suite, message structures are defined as OTDs. Each OTD consists of:

- **Physical hierarchy**

  The predefined way in which envelopes, segments, and data elements are organized to describe a particular ASC 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 an invoice that is sent from one trading partner to another defines the header, trailer, segments, and data elements required by invoice transactions. The ASC X12 OTD Library for a specific version includes transaction set structures for each of the transactions available in that version. You can use these structures as provided, or customize them to suit your business needs.

eGate Integrator uses OTDs based on ASC 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 ASC X12 transaction.

The list of transactions provided is different for each version of ASC X12.

The ASC X12 OTD Library provides ASC X12 OTDs that you can use to build Sun Java Composite Application Platform Suite Projects for interfacing with ASC X12 systems. You can use the OTDs standalone with eGate Integrator or in combination with eXchange Integrator and eGate Integrator.

2.2 ASC X12 Version Support

This product includes OTDs for the following X12 versions.

Table 2 Supported X12 Versions

▪ 4010	▪ 4020	▪ 4030	▪ 4040	▪ 4050	▪ 4060
▪ 4011	▪ 4021	▪ 4031	▪ 4041	▪ 4051	▪ 4061
▪ 4012	▪ 4022	▪ 4032	▪ 4042	▪ 4052	

The library OTDs only accept messages with all the envelope segment information. If you need to generate a custom OTD without an envelope segment, use the SEF OTD wizard as described in **“Creating ASC X12 OTDs from SEF Files” on page 26**.

2.3 SEF File Support

This product supports 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 ASC X12 OTDs from SEF Files" on page 26**.

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.

2.4 ASC X12 Validation Support

Within each X12 OTD are Java methods and Java bean nodes for handling validation (see **"performValidation" on page 44**). The marshal and unmarshal methods of the envelope OTDs handle enveloping and de-enveloping (see **"marshal" on page 43** and **"unmarshal" on page 49**). No pre-built translations are supplied with the OTD libraries; these can be built in the Java Collaboration Editor.

X12 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 35** and **"checkAll" on page 36**) is in XML format conforming to the **X12OTDErrors.xsd** file. Refer to **"Contents of the X12OTDErrors.xsd File" on page 51** for more information. For a sample of the validation output XML, refer to **"Sample of Validation Output XML" on page 52**.

2.5 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, X12 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 47**
- setMaxFreedSegsComsNum() method (**"setMaxFreedSegsComsNum" on page 47**)

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

2.6 Alternative Formats: ANSI and XML

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

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 49**.

To verify whether the output is XML, use the *getXmlOutput()* method. For information, refer to **"getXmlOutput" on page 42**.

2.6.1 XML Format for X12

Because there is XML standard for X12, the ASC X12 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*
20011001*1615*1*T*004010~ST*850*0001~BEG*01*BK*99AKDF9DAL393*39483920
193843*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*2001100
1*0718*021*20011001*1952~REF*AB*3920394930203*GENERAL
PURPOSE*BT:12345678900987654321768958473:CM:500:AB:3920394930203~PER*
AC*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.
```

2.7 Errors and Exceptions

For all X12 OTDs, including the three envelope OTDs (located in Sun > SeeBeyond > OTD Library > X12 > envelope), if 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.

The cause of the UnmarshalException depends on which envelope threw the exception:

- A com.stc.otd.runtime.UnmarshalException is thrown by X12InterchangeEnv (the X12 interchange envelope) on data with a duplicate IEA segment (or, for that matter, *any* extra segment data after a regular IEA segment) or a duplicate inner level GE segment, but not on data with a missing IEA segment or a missing inner level GE segment.
- A com.stc.otd.runtime.UnmarshalException is thrown by "X12FunctionalGroupEnv (the X12 functional group envelope) on data with a duplicate GS segment (or, for that matter, *any* extra segment data after a regular GS segment), but not on data with a missing GS segment or a missing transaction set level SE segment, and not on data with one or more duplicate transaction set level SE segments.
- A com.stc.otd.runtime.UnmarshalException is thrown by X12TransactionSetEnv (the X12 transaction set envelope) on data with one or more duplicate SE segments.

You can also use the *isUnmarshalComplete()* method on an OTD to verify whether the *unmarshal()* method completed its parsing 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, see **"On Demand Parsing" on page 13**. Encountering this exception 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.

2.8 ISA Segment Parsing

The ISA segment in an ASC X12 message is of fixed length, and has sixteen fixed-length data fields. The fixed-length property of the ISA segment allows you to define the fixed positions for delimiters, such as segment terminator and element separator, that are critical for parsing the X12 message. Logic is built into the fully-enveloped OTDs (including the X12 Interchange Envelope OTD) to retrieve the delimiters at certain fixed positions and use these delimiters to unmarshal the message.

However, if any of the sixteen data fields is longer or shorter than defined in the specification, the positions of delimiters are nonstandard. This will result in parsing difficulties and probable runtime failure. In some cases, data that is mis-specified in this way can cause conflicting delimiter values (such as element separator defined the same as segment terminator), rendering the whole message unparsable.

Steps Taken to Check for ISA Segment Errors

Assuming valid data, the logic for checking ISA segment errors is as follows:

1. Check whether the character at index=3 is a valid element separator. If not, an UnmarshalException is thrown, and the unmarshaling process is terminated.
2. Check whether the characters at index=6,17,20,31,34,50,53,69,76,81,83,89,99,101,103 is same as that at index=3. If not, an UnmarshalException is thrown with an error message that begins "**ISA Segment Error**", and processing continues with variable length parsing at step 6.
3. Check whether the character at index=104 is a valid subelement separator. If not, an UnmarshalException is thrown with an error message that begins "**ISA Segment Error**", and processing continues with variable-length parsing at step 6.
4. Check whether the character at index=105 is a valid segment terminator, and also verify that this character is different from the element separator and subelement separator. If not, an UnmarshalException is thrown with an error message that begins "**ISA Segment Error**", and processing continues with variable-length parsing at step 6.
5. If the version of X12 is version 4020 or later, check whether the character at index=82 is a valid repetition separator and different from the element separator, subelement separator, and segment terminator. If not, an UnmarshalException is thrown with an error message that begins "**ISA Segment Error**", and processing continues with variable-length parsing at step 6.
6. If variable-length parsing becomes necessary, the ISA segment is parsed using the element separator determined at step 1 to retrieve the next sixteeen data elements. After this ISA parsing, an UnmarshalException is thrown with an error message that begins "**ISA Segment Error**", and the OTD with ISA segment parsed may be examined later to retrieve the parsed ISA segment.

If the message does not have a terminator for its last segment terminator, the ISA segment is also parsed before an UnmarshalException is thrown. You can catch this UnmarshalException and examine the parsed ISA segment in the OTD (to generate negative TA1 acknowledgment, for example).

Chapter 3

Installing the ASC X12 OTD Library

This chapter describes how to install the X12 OTD Library and its documentation. This chapter also includes the system requirements and supported operating systems for the ASC X12 OTD Library.

What's in This Chapter

3.1 System Requirements

Each X12 OTD **.sar** file requires approximately 8 MB disk space; the combined disk space required to load all **.sar** files is approximately 140 MB.

Due to the size of the X12 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 20**.

Other than that, the system requirements for the X12 OTD Library are the same as those for eGate Integrator and eXchange Integrator. For information, refer to the *Sun Java Composite Application Platform Suite Installation Guide*.

3.2 Supported Operating Systems

For information about supported operating systems, refer to the B2B Readme file.

3.3 Installing the ASC X12 OTD Library

During the ASC X12 OTD Library installation process, the Enterprise Manager, a Web-based application, is used to select and upload products as **.sar** files from the Sun Java Composite Application Platform 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 eGate Integrator in the *Sun Java Composite Application Platform Suite Installation Guide*, and include the steps below to install the ASC X12 OTDs. You must have uploaded a **Product_List.sar** to the CAPS Repository that includes a license for the ASC X12 OTD Library.

To install the ASC X12 OTD Library

1. After uploading the **eGate.sar** file to the eGate Repository, select and upload the items below as described in the *Sun Java Composite Application Platform Suite Installation Guide*:
   - The **.sar** file for the OTDs you want to install, such as **X12_v4050_OTD.sar** (to install version 4050)
   - **X12_OTD_Docs.sar** (to install the user's guide)
   - Optionally, also upload **SEF_OTD_Wizard.sar** (to install the SEF OTD wizard; this allows you to build ASC X12 OTDs from SEF files)
   - Optionally, also upload **eXchange.sar, X12_Manager.sar**, and the ***_Validation_BP.sar** file corresponding to each OTD library you upload, such as **X12_v4050_OTD_Validation_BP.sar**.
2. Click the **DOCUMENTATION** page, click **ASC X12 OTD Library** in the left pane, and click **Sun B2B Suite ASC X12 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**.

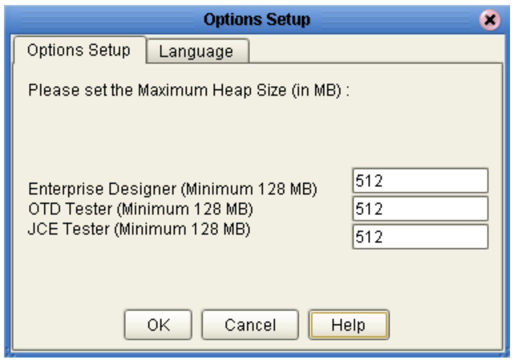
3.4 Increasing the Enterprise Designer Heap Size

Due to the size of the ASC X12 OTDs, you may need to increase the heap size property of 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 Enterprise Designer, OTD Tester, and JCE Tester to no less than 512 MB, and click **OK**.

Figure 1 Increasing Enterprise Designer Heap Size



Please set the Maximum Heap Size (in MB) :

Enterprise Designer (Minimum 128 MB) 512
OTD Tester (Minimum 128 MB) 512
JCE Tester (Minimum 128 MB) 512

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 **<*jcaps51*>\edesigner\bin**, where **<*jcaps51*>** is the folder where Enterprise Designer 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 Enterprise Designer.

Chapter 4

Using ASC X12 OTDs

This chapter describes how you use ASC X12 OTDs provided in the ASC X12 OTD Library, such as customizing OTDs and building ASC X12 Collaborations.

What's in This Chapter

4.1 Displaying ASC X12 OTDs

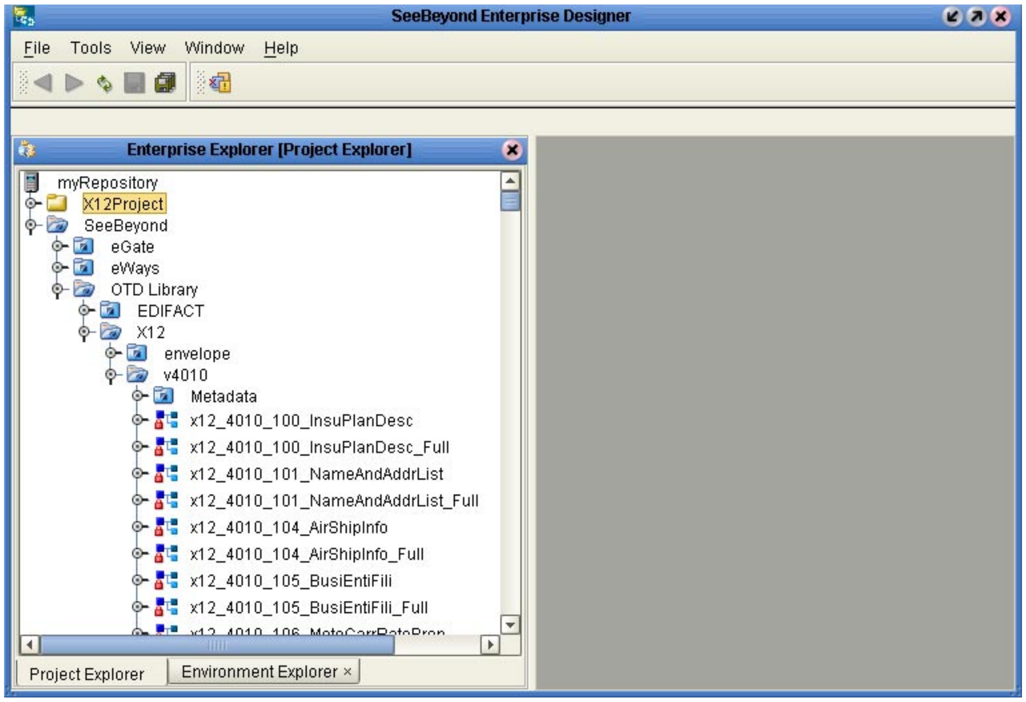
After installing the ASC X12 OTDs, you can view the OTDs in the OTD Editor as described below.

To display ASC X12 OTDs

1. In the **Project Explorer** tab of Enterprise Designer, expand the following folders:
   - **SeeBeyond**
   - **OTD Library**
   - **X12**
   - Folder containing the X12 version

   The Project Explorer tab displays the installed OTDs per ASC X12 version, for example **v4010**.

Figure 2 OTDs for ASC X12 Version 4010


SeeBeyond Enterprise Designer
File Tools View Window Help
Enterprise Explorer [Project Explorer]
myRepository
X12Project
SeeBeyond
eGate
eWays
OTD Library
EDIFACT
X12
envelope
v4010
Metadata
x12_4010_100_InsuPlanDesc
x12_4010_100_InsuPlanDesc_Full
x12_4010_101_NameAndAddrList
x12_4010_101_NameAndAddrList_Full
x12_4010_104_AirShipInfo
x12_4010_104_AirShipInfo_Full
x12_4010_105_BusiEntiFili
x12_4010_105_BusiEntiFili_Full
Project Explorer
Environment Explorer

The **Project Explorer** tab displays the OTDs available for the ASC X12 version folder selected. The table below described the OTD naming conventions.

Table 3 OTD Naming Convention

x12_	Protocol name
v*nnnn*_	ASC X12 version
997_FuncAckn	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 the SEF file for the OTDs. You can use the SEF file to customize the OTD as described in **Customizing the ASC X12 OTDs** on page 25.

4.2 Building ASC X12 OTD Collaborations

This section describes how you build Java Collaborations that use the ASC X12 OTDs provided in the ASC X12 OTD Library.

To customize the OTDs before building the Collaboration, refer to **"Customizing the ASC X12 OTDs" on page 25**.

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 ASC X12 OTD Library" on page 19**.

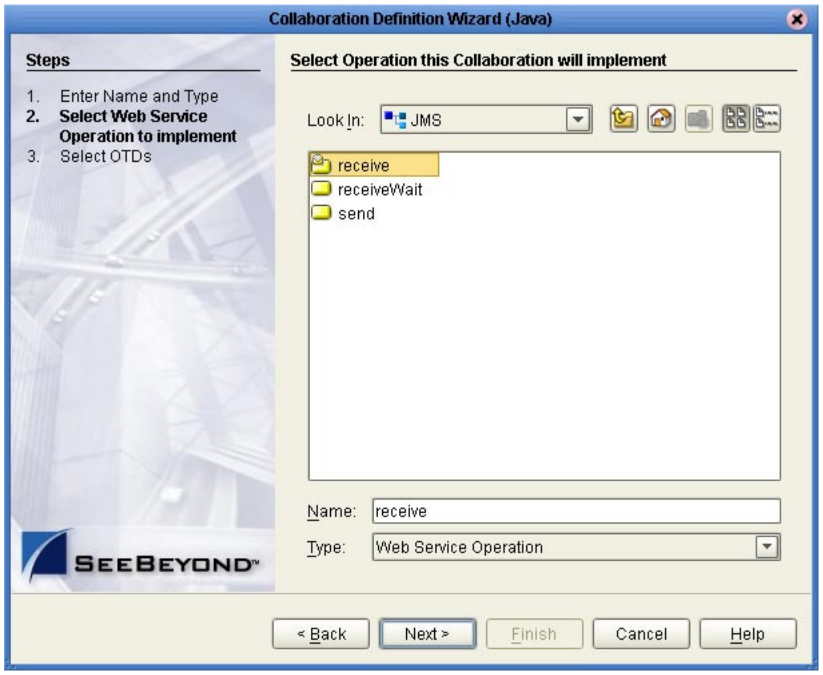
To build ASC X12 OTD Collaborations

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


Collaboration Definition Wizard (Java)
Steps
1. Enter Name and Type
2. Select Web Service Operation to implement
3. Select OTDs
Select Operation this Collaboration will implement
Look In:
JMS
receive
receiveWait
send
SEEBEYOND
< Back
Next >
Finish
Cancel
Help

Name: receive
Type: Web Service Operation

The **Select OTDs** page appears.

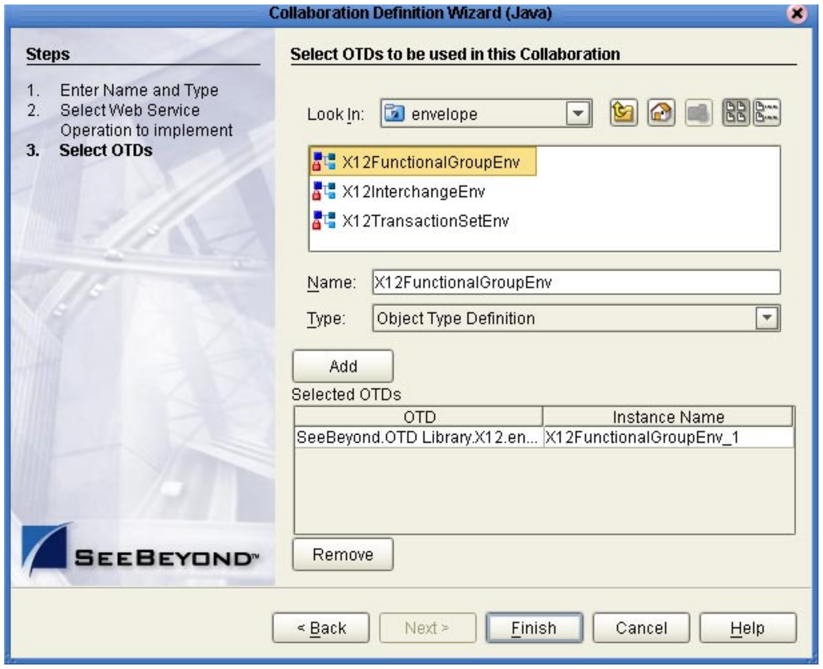
4 To use envelopes OTDs, under **Look In,** navigate to the envelopes by double-clicking the folders below. If the Collaboration does not use enveloping, continue with step 6.

- **SeeBeyond**
- **OTD Library**
- **X12**
- envelope

The **Look In** area displays the envelope OTDs.

5 Double-click the envelope(s) to be used. This adds the envelopes under **Selected OTDs**.

Figure 4 Adding Envelopes to the Collaboration


Collaboration Definition Wizard (Java)
Steps
1. Enter Name and Type
2. Select Web Service Operation to implement
3. Select OTDs
Select OTDs to be used in this Collaboration
SeeBeyond
Add
Remove
< Back
Next >
Finish
Cancel
Help

Look In: envelope

X12FunctionalGroupEnv
X12InterchangeEnv
X12TransactionSetEnv

Name: X12FunctionalGroupEnv

Type: Object Type Definition

Selected OTDs

OTD	Instance Name
SeeBeyond.OTD Library.X12.en...	X12FunctionalGroupEnv_1

6 Under **Look In,** navigate to the OTDs by double-click the following folders:

- **SeeBeyond**
- **OTD Library**
- **X12**
- Folder indicating the ASC X12 version, such as **v4010**

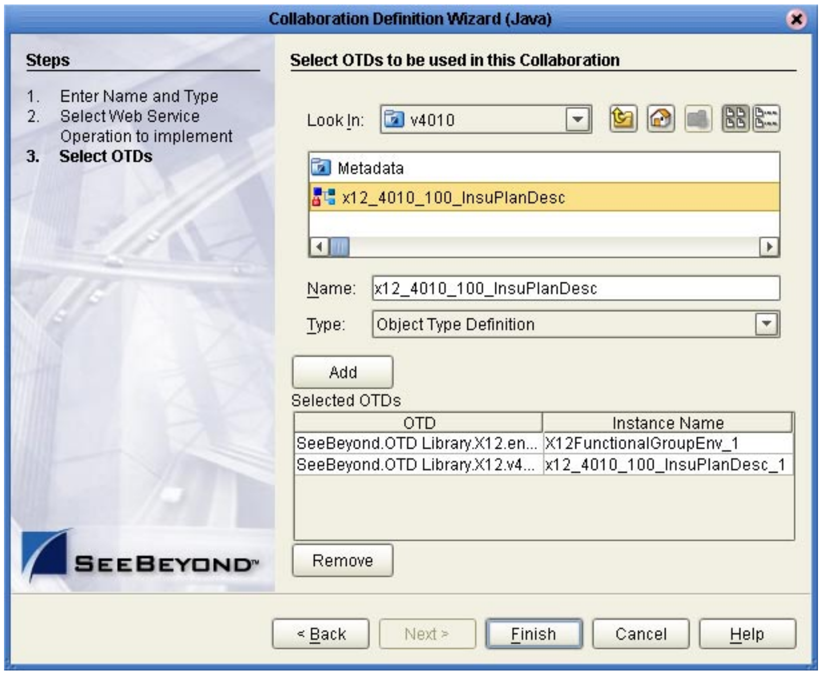
The **Look In** area displays the OTDs for the selected ASC X12 directories. The table below describes the naming convention for the OTDs.

Table 4 OTD Naming Convention

x12_	Protocol name
v*nnnn*_	ASC X12 version
997_FuncAckn	Transaction code and transaction name abbreviation
_Full	Fully enveloped OTD version that includes the inner and outer envelopes

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

Figure 5 Adding OTDs to the Collaboration


Collaboration Definition Wizard (Java)
Steps
1. Enter Name and Type
2. Select Web Service Operation to implement
3. Select OTDs
Select OTDs to be used in this Collaboration
Look In:
v4010
Metadata
x12_4010_100_InsuPlanDesc
Name:
x12_4010_100_InsuPlanDesc
Type:
Object Type Definition
Add
Selected OTDs
Remove
SEEBEYOND
< Back
Next >
Finish
Cancel
Help

OTD	Instance Name
SeeBeyond.OTD Library.X12.en...	X12FunctionalGroupEnv_1
SeeBeyond.OTD Library.X12.v4...	x12_4010_100_InsuPlanDesc_1

8 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 ASC X12 OTD methods, refer to **"Java Methods for ASC X12 OTDs" on page 30**.

4.3 Customizing the ASC X12 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 ASC 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 file locally for editing.

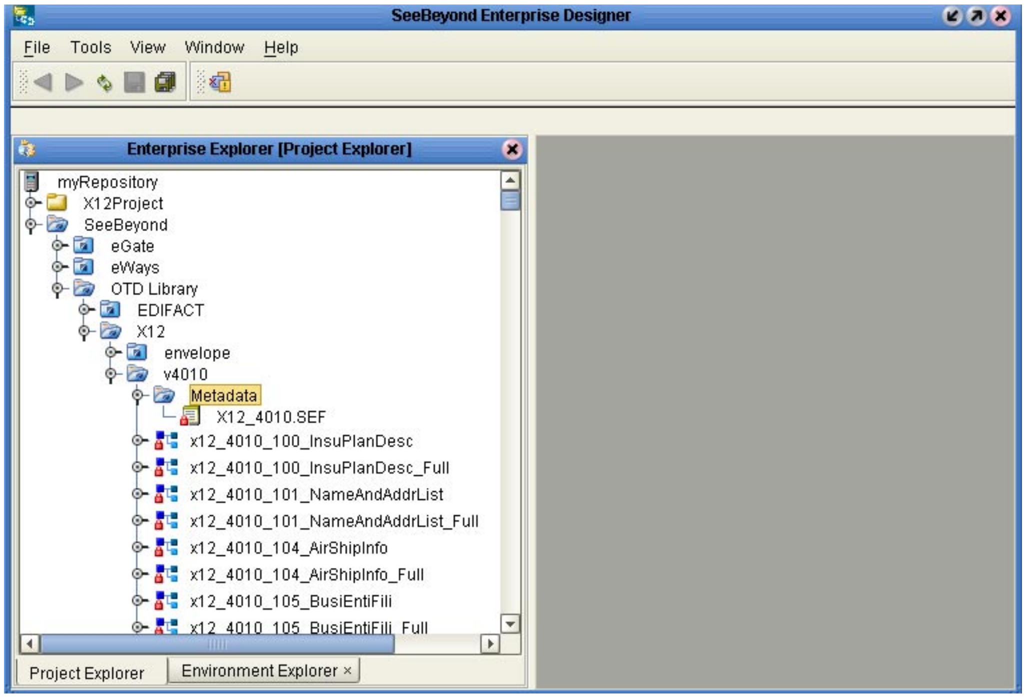
To customize ASC X12 OTDs

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

- **SeeBeyond**
- **OTD Library**
- **X12**
- Folder indicating the ASC X12 version, such as **v4010**
- **Metadata**

The metadata folder displays the available SEF file.

Figure 6 Saving ASC X12 OTD


SeeBeyond Enterprise Designer
File Tools View Window Help
Enterprise Explorer [Project Explorer]
myRepository
X12Project
SeeBeyond
eGate
eWays
OTD Library
EDIFACT
X12
envelope
v4010
Metadata
X12_4010.SEF
x12_4010_100_InsuPlanDesc
x12_4010_100_InsuPlanDesc_Full
x12_4010_101_NameAndAddrList
x12_4010_101_NameAndAddrList_Full
x12_4010_104_AirShipInfo
x12_4010_104_AirShipInfo_Full
x12_4010_105_BusiEntiFili
x12_4010_105_BusiEntiFili_Full
Project Explorer
Environment Explorer

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

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

4 Use a SEF editor to customize the SEF file.

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

4.4 Creating ASC X12 OTDs from SEF Files

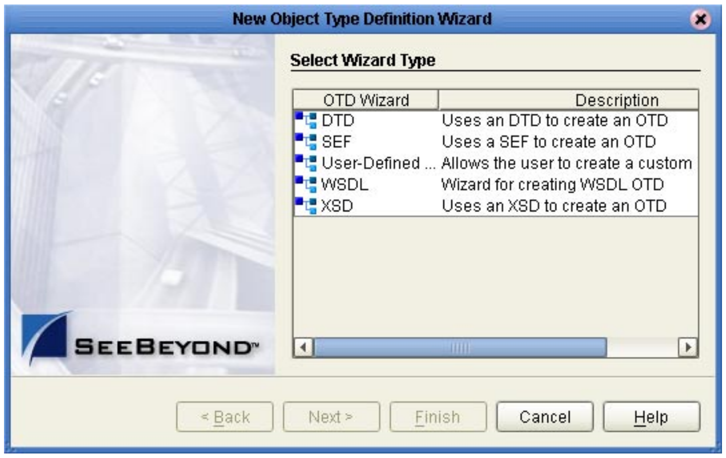
This section describes how you create ASC X12 OTDs using SEF files. The ASC X12 OTD Library includes the SEF files for the OTDs to allow you to customize the OTD as described in the section above. Once you have tailored the SEF file to your business requirements, you can then use the procedure below 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 eGate Repository, and used the **Update Center** in Enterprise Designer to install it. For information, refer to **"Installing the ASC X12 OTD Library" on page 18**.

To create ASC X12 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 7 Creating ASC X12 OTDs

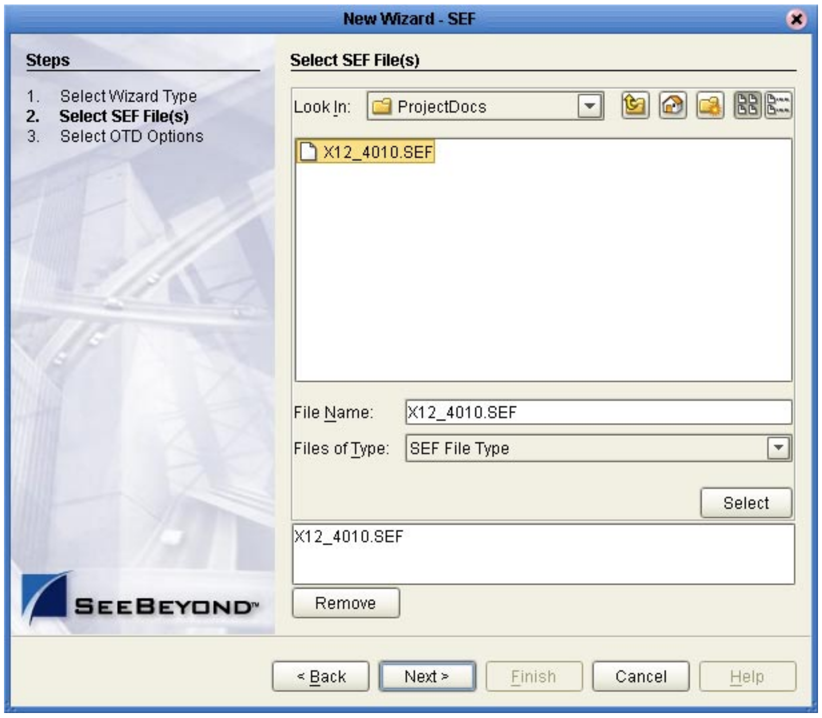


OTD Wizard	Description
DTD	Uses an DTD to create an OTD
SEF	Uses a SEF to create an OTD
User-Defined ...	Allows the user to create a custom
WSDL	Wizard for creating WSDL OTD
XSD	Uses an XSD to create an OTD

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 8 Selecting the SEF File



Look In: ProjectDocs

X12_4010.SEF

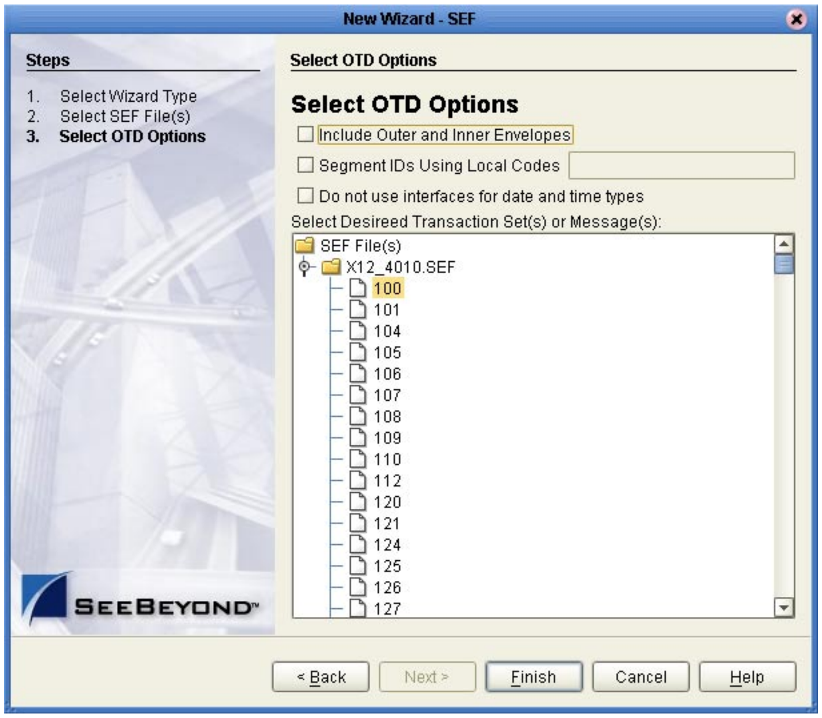
File Name: X12_4010.SEF

Files of Type: SEF File Type

X12_4010.SEF

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

Figure 9 Selecting the OTD Options



Select OTD Options

Select OTD Options

☐ Include Outer and Inner Envelopes

☐ Segment IDs Using Local Codes

☐ Do not use interfaces for date and time types

Select Desireed Transaction Set(s) or Message(s):

```
SEF File(s)
  X12_4010.SEF
    100
    101
    104
    105
    106
    107
    108
    109
    110
    112
    120
    121
    124
    125
    126
    127
```

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, 070215 in the input file might be represented as 20070215 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 ASC X12 OTDs

This chapter describes the Java methods available for ASC X12 OTDs.

What's in This Chapter

5.1 Get and Set Methods

The OTDs in the ASC X12 OTD Library contain the Java methods that enable you to set and get the delimiters, which in turn extend the functionality of the ASC X12 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 45
- **getElementSeparator** on page 38 and **setElementSeparator** on page 46
- **getFGValidationResult** on page 38
- **getICValidationResult** on page 38
- **getInputSource** on page 38
- **getMaxDataError** on page 39 and **setMaxDataError** on page 47
- **getMaxFreedSegsComsNum** on page 39 and **setMaxFreedSegsComsNum** on page 47
- **getMaxParsedSegsComsNum** on page 40and **setMaxParsedSegsComsNum** on page 47
- **getMsgValidationResult** on page 40
- **getRepetitionSeparator** on page 40 and **setRepetitionSeparator** on page 47
- **getSegmentCount** on page 41
- **getSegmentTerminator** on page 41 and **setSegmentTerminator** on page 48
- **getSubelementSeparator** on page 41 and **setSubelementSeparator** on page 48

- **getTSValidationResult** on page 42
- **getUnmarshalError** on page 42
- **getXmlOutput** on page 42

The following methods are available from the loop elements:

- **getLoopxxx** on page 39 and **setLoopxxx** on page 46
- **getSegmentCount** on page 41
- **setXmlOutput** on page 49

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 ASC X12 delimiters are as follows:

- Data element separator (default is an asterisk)
- Subelement separator/component element separator (default is a colon)
- Repetition separator (version 4020 and later) (default is a plus sign)
- Segment terminator (default is a tilde)

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 Messages

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.

Setting Delimiters

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 45
- **setElementSeparator** on page 46
- **setSegmentTerminator** on page 48
- **setSubelementSeparator** on page 48
- **setRepetitionSeparator** on page 47
- **setSubelementSeparator** on page 48)

If the input data is already unmarshaled into an ASC X12 OTD, you can use the get methods to retrieve the delimiters from the input data. For information, refer to **"Get and Set Methods" on page 30**.

5.2.2 Outgoing Messages

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 **"Setting Delimiters" on page 31**.)

Note that the interchange-level object is called a *fully-enveloped* OTD; the message-level object is called a *non-enveloped* OTD.

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

To add the support of serialization (i.e. marshalling) of non-root level objects (segments, composites, and segment loops) in addition to root level objects (i.e. interchange level objects, group level objects and message level objects), each of the non-root level objects now contains a set of delimiters appropriate to its message type which are accessible through a few methods/APIs mentioned below to the user.

Delimiter Set

The delimiter set for the ANSI X12 message type consists of the following:

- segmentTerminator
- elementSeparator
- subelementSeparator
- repetitionSeparator
- segmentCodeSeparator (equivalent to elementSeparator)
- decimalMark

Accessor Methods

The methods to access the delimiters inside a root-levl or non-root-level object are:

- **getSegmentTerminator**
- **setSegmentTerminator**

- **getElementSeparator**
- **setElementSeparator**
- **getSubelementSeparator**
- **setSubelementSeparator**
- **getRepetitionSeparator**
- **setRepetitionSeparator**
- **getSegmentCodeSeparator**
- **setSegmentCodeSeparator**
- **getDecimalMark**
- **setDecimalMark**
- **setDefaultDelimiters**

Initialization of the Delimiter Set

The delimiters of a root-level or non-root-level object are defaulted to their industry standard values when the delimiter set is created. Each individual delimiter can be changed by the corresponding setter method during initialization or later invocation by the user.

The initialization of the delimiter set can be triggered either by any of the **"Accessor Methods" on page 32**, or by any the following additional methods:

- *All segments:* **marshalToBytes**
- *ISA segment only:* **setEI10_11_InteContStanIden** (X12 version 4010 or lower, or interchange envelope OTD)
- *ISA segment only:* **setEI65_11_RepeSepa** (X12 version 4020 or higher)

Precedence of Delimiters

When a fully-enveloped OTD or interchange envelope OTD is used to marshal its content into an outgoing message, the delimiter values in the first delimiter field-containing segment (the ISA segment) can sometimes conflict with the delimiter values specified at the interchange level (that is, at the OTD level). This occurs because the interchange level objects and non-root-level objects can separately allow a user to set delimiters independently in a fully-enveloped OTD.

The delimiter values in the ISA segment, if initialized, take precedence over the delimiter values set for the fully-enveloped OTD or interchange envelope OTD. The precedence order can therefore be represented as follows:

Delimiter set in the ISA segment (if initialized) > Delimiter set in the OTD

Example Showing Delimiter Precedence

The following example method illustrates the precedence of delimiters set in the X12 ISA segment over delimiters set in the root level “com.stc.x12env.runtime.ic.ICEnv” object in an X12 interchange envelope OTD:

```
public String generateOutput() throws Exception {

     String encoding = "utf-8";

    // (1) Create a new instance of X12 Interchange Envelope OTD
    com.stc.x12env.runtime.ic.ICEnv icEnvOtd =
new com.stc.x12env.runtime.ic.ICEnv();

    // (2) Set delimiters in the Interchange Envelope OTD
    icEnvOtd.setSegmentTerminator('~');
    icEnvOtd.setElementSeparator('+');
    icEnvOtd.setSubelementSeparator('^');

    // (3) Create a new ISA segment object
    com.stc.x12env.runtime.ic.ISA isaSegment =
new com.stc.x12env.runtime.ic.ISA();

    // (4) Populate the fields inside the ISA
    isaSegment.setEI01_1_AuthInfoQual("00");
    isaSegment.setEI02_2_AuthInfo("          ");
    isaSegment.setEI03_3_SecuInfoQual("01");
    isaSegment.setEI04_4_SecuInfo("          ");
    isaSegment.setEI05_5_InteIDQual("13");
    isaSegment.setEI06_6_InteSendID("3105451234");
    isaSegment.setEI05_7_InteIDQual("16");
    isaSegment.setEI07_8_InteReceID("123456789");
    com.stc.runtime.dt.Date date =
com.stc.otd.runtime.edi.EdiDate.parse8("20070115");
    isaSegment.setEI08_9_InteDate(date);
    com.stc.runtime.dt.Time time =
com.stc.otd.runtime.edi.EdiTime.parse4("1647");
    isaSegment.setEI09_10_InteTime(time);
    isaSegment.setEI10_11_InteContStanIden("U");
    isaSegment.setEI11_12_InteContVersNumb("00301");
    isaSegment.setEI12_13_InteContNumb(905);
    isaSegment.setEI14_15_UsagIndi("T");
    isaSegment.setEI13_14_AcknRequ("1");
    isaSegment.setEI15_16_CompElemSepa(":");

    // (5) Set the rest of delimiters inside the ISA segment object
    isaSegment.setSegmentTerminator('!');
    isaSegment.setElementSeparator('*');

    // (6) Set the populated ISA segment object to the Interchange
    //     Envelope OTD
    icEnvOtd.setISA_InteContHead(isaSegment);

    // (7) Get the IEA segment object inside the Interchange
    //     Envelope OTD; also creates the IEA segment instance
    com.stc.x12env.runtime.ic.IEA ieaSegment =
icEnvOtd.getIEA_InteContTrai();

    // (8) Populate the fields inside the IEA
    ieaSegment.setEI12_2_InteContNumb(905);
    ieaSegment.setEI16_1_NumbOfInclFuncGrou(1);
```

```
        // (9) Provide the Functional Group data
        String funcGrp = "
    GS*FA*123*321*927003*1203*1112*T*004010!ST*997*0001!AK1*FA*1!AK2*9
    97*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*1112!";

        // (10) Set the functional group data inside the Interchange
        //      Envelope OTD
         icEnvOtd.setFunctionalGroup(0, funcGrp.getBytes(encoding));

        // (11) Invoke API to generate serialized byte array output
        //      of the Interchange Envelope OTD
        byte[] results = icEnvOtd.marshalToBytes();

        // (12) return as a string
        return new String(results, encoding);

         }
```

The foregoing example method returns the following string as output:

```
ISA*00*          *01*          *13*3105451234     *16*123456789
*051215*1647*U*00301*000000905*1*T*:!
GS*FA*123*321*927003*1203*1112*T*004010!ST*997*0001!AK1*FA*1!AK2*9
97*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*1112!IEA*1*000000905!
```

5.3 Available Methods

This section describes the signature and description for each available ASC X12 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 available only for full-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.

getDecimalMark

Signature

```
public char getDecimalMark()
```

Description

Returns the decimal mark.

Exceptions

None.

getElementSeparator

Signature

```
public char getElementSeparator()
```

Description

Gets the elementSeparator character.

Exceptions

None.

Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
char elmSep=myOTD.getElementSeparator();
```

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 for fully enveloped OTDs at the Outer and Inner root levels.

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 root level 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 13**.

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 13**.

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 44**.

This method is available only at the Outer, Inner and transaction set levels in fully enveloped OTDs.

Exceptions

None.

getRepetitionSeparator

Signature

```
public char getRepetitionSeparator()
```

Description

Returns the repetition separator character.

Exceptions

None.

Example

```
x12_4020.x12_4020_850_PurcOrde_Outer myOTD=new x12_4020.x12_4020_850_
PurcOrde_Outer();
......
......
char repSep=myOTD.getRepetitionSeparator();
```

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.

Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
char segTerm=myOTD.getSegmentTerminator();
```

getSubelementSeparator

Signature

```
public char getSubelementSeparator()
```

Description

Returns the subelement/composite element separator character.

Exceptions

None.

Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
char subeleSep=myOTD.getSubelementSeparator();
```

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 methods is available at the Outer, Inner, and transaction set levels in fully enveloped OTDs. It is also available at the top 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 14**.

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 13** and **"Errors and Exceptions" on page 16**.

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 **"ASC X12 Validation Support" on page 13**.

Exceptions

None.

Example

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

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 set`*`xxx`*`(`*`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 ASC X12 delimiters:

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

For more information, refer to **“Setting Delimiters” on page 31**.

Exceptions

None.

Example

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

setElementSeparator

Signature

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

Description

Sets the element separator character. For more information, refer to **“Setting Delimiters” on page 31**.

Exceptions

None

Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
char c='+';
myOTD.setElementSeparator(c);
```

setLoopxxx

Signature

`public void set`*`Loopxxx`*`(item)`

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

`public void set`*`Loopxxx`*`(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 13**.

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 13**.

Exceptions

None.

setRepetitionSeparator

Signature

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

Description

Sets the repetition separator character. For more information, refer to **"Setting Delimiters" on page 31**.

Exceptions

None.

Example

```
x12_4030.x12_4030_850_PurcOrde_Outer myOTD=new x12_4030.x12_4030_850_
PurcOrde_Outer();
......
......
char c='*';
myOTD.setRepetitionSeparator(c);
```

setSegmentTerminator

Signature

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

Description

Sets the segment terminator character. For more information, refer to **"Setting Delimiters" on page 31**.

Exceptions

None.

Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
char c='~';
myOTD.setSegmentTerminator(c);
```

setSubelementSeparator

Signature

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

Description

Sets the subelement separator character. For more information, refer to **"Setting Delimiters" on page 31**.

Exceptions

None.

Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
char c=':';
myOTD.setSubelementSeparator(c);
```

setXmlOutput

Signature

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

Description

When used with the parameter set to **true**, this method causes the X12 OTD involved to output XML.

When used with the parameter set to **false**, this method causes the X12 OTD to output ANSI (which is the default output if this method is not used at all).

For more information, refer to **"Alternative Formats: ANSI and XML" on page 14**.

Exceptions

None.

Example

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

unmarshal

Signature

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

Description

Unmarshals the given input into an internal data tree.

For more information **"On Demand Parsing" on page 13** and **"Errors and Exceptions" on page 16**.

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 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 XML schema file the validation output string conforms to. This appendix also includes a sample of validation output XML. For more information, refer to **"ASC X12 Validation Support" on page 13** and **"performValidation" on page 44**.

What's in This Appendix

A.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 X12 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>
```

A.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