SeeBeyond ICAN Suite

ASC X12 OTD Library User's Guide

Release 5.0.4

Version 20041118113416.

Contents

List of Figures

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

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 document contains the following information:

- **Chapter 1, "Introduction"**, provides a general preview of this document, its purpose, scope, and organization.
- **Chapter 2, "Overview of the 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 XML.

1.1.2 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.

1.1.3 Intended Audience

This document provides information for those who are designing, deploying, and managing ICAN Projects that use X12 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 System Administrator Guide*

- *eXchange Integrator User's Guide*

1.2 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

1.3 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 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 About the ASC X12 OTD Library

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

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

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

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

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

X12 implementation guides can be obtained from Washington Publishing Company:

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

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 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 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 Object Type Definitions 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 ICAN 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 Directory Support

The X12 OTD library 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 24**.

2.3 SEF File Support

The ASC X12 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 ASC X12 OTDs from SEF Files" on page 24**.

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 37**). The marshal and unmarshal methods of the envelope OTDs handle enveloping and de-enveloping (see **"marshal" on page 37** and **"unmarshal" on page 42**). 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 29** and **"checkAll" on page 29**) is in XML format conforming to the **X12OTDErrors.xsd** file. Refer to **"Contents of the X12OTDErrors.xsd File" on page 44** for more information. For a sample of the validation output XML, refer to **"Sample of Validation Output XML" on page 45**.

2.5 On Demand Parsing

For performance enhancement reasons, the **unmarshal()** method does not unmarshal the entire message. 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 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 in the Collaboration using a **get*xxx*()** method. The OTD may assigned 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 40**
- setMaxFreedSegsComsNum() method (**"setMaxFreedSegsComsNum" on page 40**)

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 change the existing Business Processes or Collaborations.

By default, the OTD output is ANSI. To change the output to XML, use the following OTD methods: However, there are two Java Methods available for setting the output to XML:

- setXMLOutput (boolean isXML)

  If the Business Process or Collaboration is set to automatically publish (the default), set the argument to true to automatically publish XML output. For information, refer to **"setXmlOutput" on page 42**.

- marshal (boolean isXMLOutput)

  If the Collaboration is set to manual publication (via the **Manual Publish** check box in the Collaboration Rules component), set the argument to true to manually publish XML output. For more information, refer to **"marshal" on page 37**.

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 two envelope OTDs, if the incoming message cannot be parsed (for example, if the OTD cannot find the UNB segment), then the **unmarshal()** method generates a com.stc.otd.runtime.UnmarshalException.

You can also use the **isUnmarshalComplete()** method to learn whether **unmarshal()** executed without reporting any errors. Successful completion does not guarantee that the OTD instance is free of unmarshal exceptions within segments, however, since elements are not unmarshaled until the first **getElement*Xxxx*()** method of a segment is encountered (see **"On Demand Parsing" on page 12**). Encountering this triggers an automatic background unmarshal of the entire segment. Note that the value returned by **isUnmarshalComplete()** 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 ASC X12 OTD Library

This chapter describes how to install the X12 OTD Library and its documentation.

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 the Enterprise Designer. For information, refer to **"Increasing the Enterprise Designer Heap Size" on page 18**.

Other than that, the system requirements for the X12 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 ASC X12 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 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 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 ASC X12 OTDs. You must have uploaded a **license.sar** to the ICAN Repository that includes a license for the ASC X12 OTD Library.

To install the ASC X12 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*:
   - The **.sar** file for the OTDs to be used, for example **ASC_X12_OTD_Lib_v4050.sar** (to install version 4050)
   - **X12_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 **ASC X12 OTD Library** in the left pane, and click **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**.

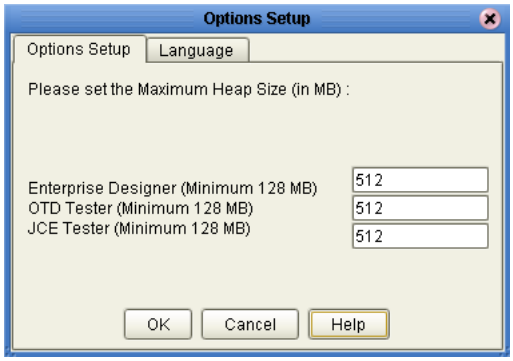
3.4 Increasing the Enterprise Designer Heap Size

Due to the size of the ASC X12 OTDs, you may need to increase the heap size property of the 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 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.

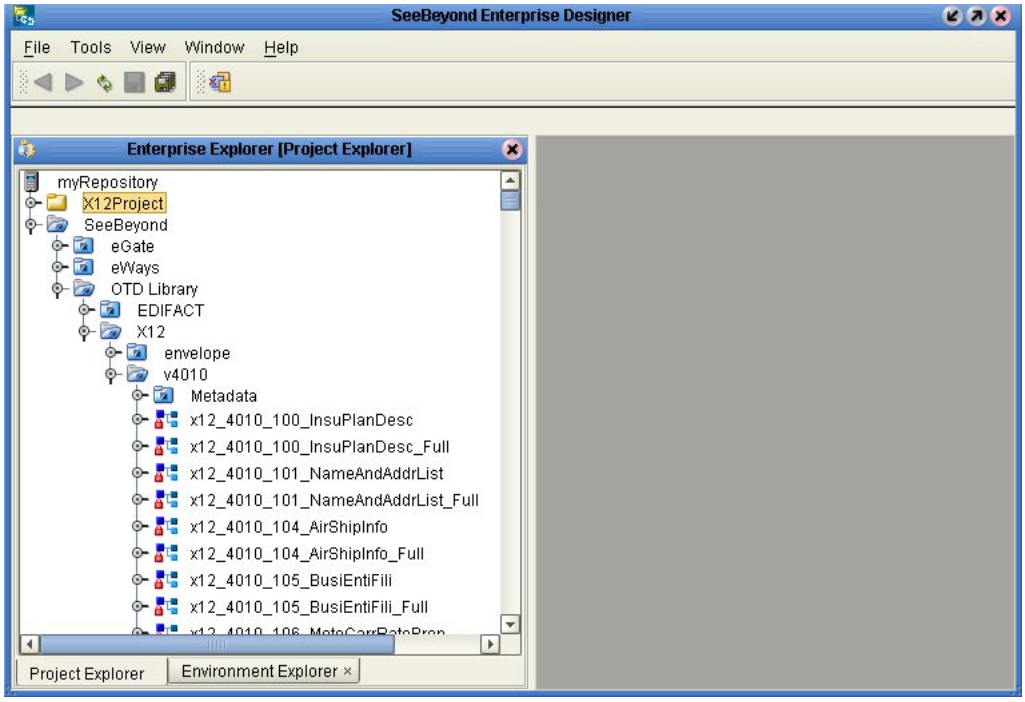
To display ASC X12 OTDs

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



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
vnnnn_	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 files for the OTDs. You can use the SEF files to customize the OTD as described in **Customizing the ASC X12 OTDs** on page 23.

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.

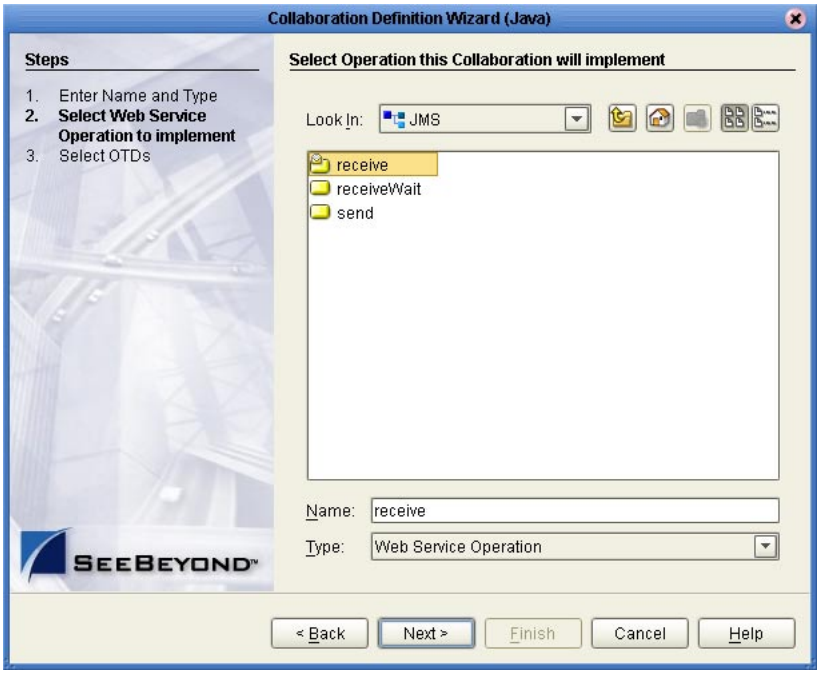
To customize the OTDs before building the Collaboration, refer to **"Customizing the ASC X12 OTDs" on page 23**.

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 17**.

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

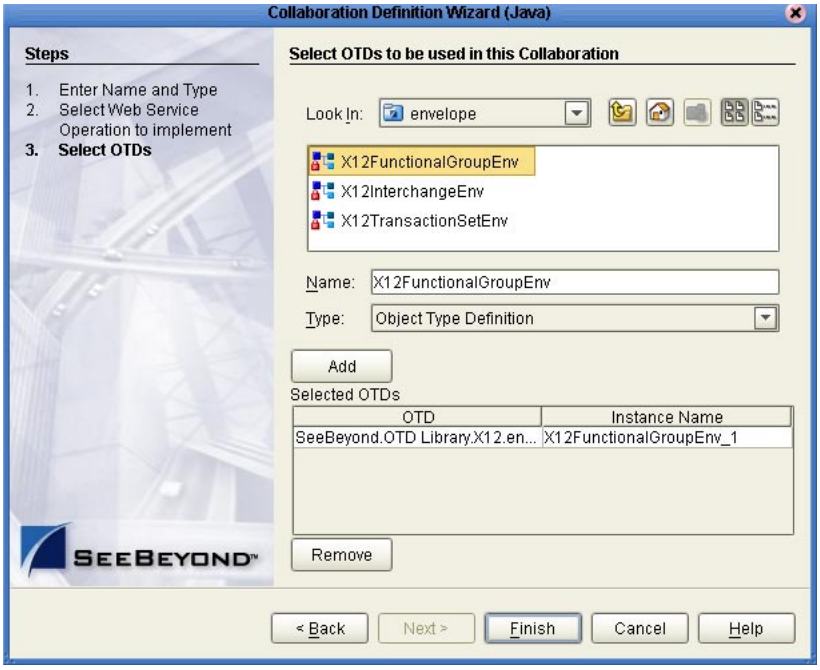


The **Select OTDs** page appears.

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



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**

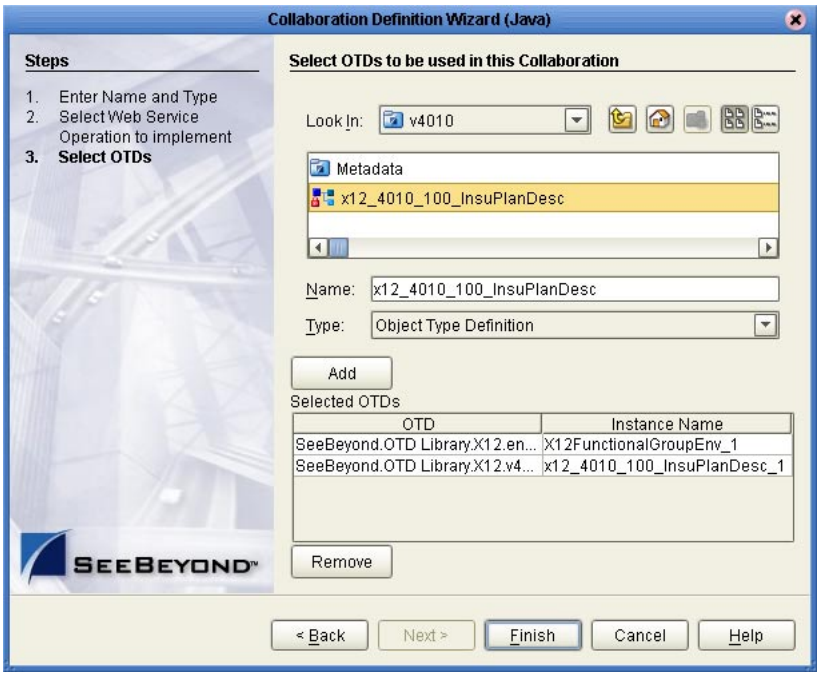
The **Look In** area displays the OTDs for the selected ASC X12 directories. The table below describes the naming convention for the OTDs.

Table 4 OTD Naming Convention

x12_	Protocol name
***vnnnn*_**	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



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 27**.

4.3 Customizing the ASC X12 OTDs

OTDs provided in the OTD Library cannot be customized. However, the OTD Library provides the SEF files to allow you to modify the file and then rebuild it. 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.

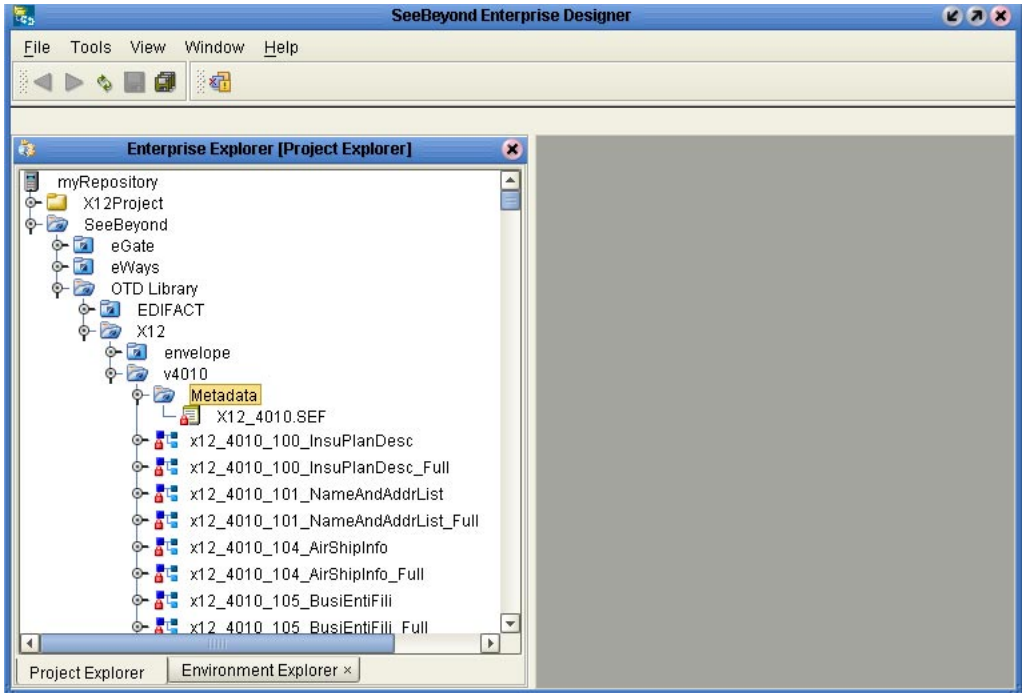
To customize ASC X12 OTDs

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 SEF files available.

Figure 6 Saving ASC X12 OTD SEF Files



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 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

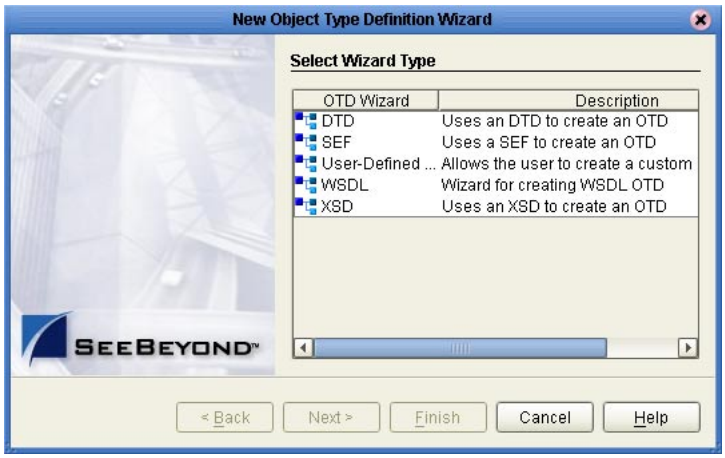
This section describes how you create ASC X12 OTDs using SEF files. The ASC X12 OTD Library includes the SEF files for the OTDs to allow you to customize the OTD as described in the section above. Once you have tailored the SEF file to your business requirements, you can then use the procedure below to recreate the OTD.

To create OTDs from SEF files, you use the SEF OTD wizard to build the OTD using a selected SEF file. 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 ASC X12 OTD Library" on page 16**.

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.

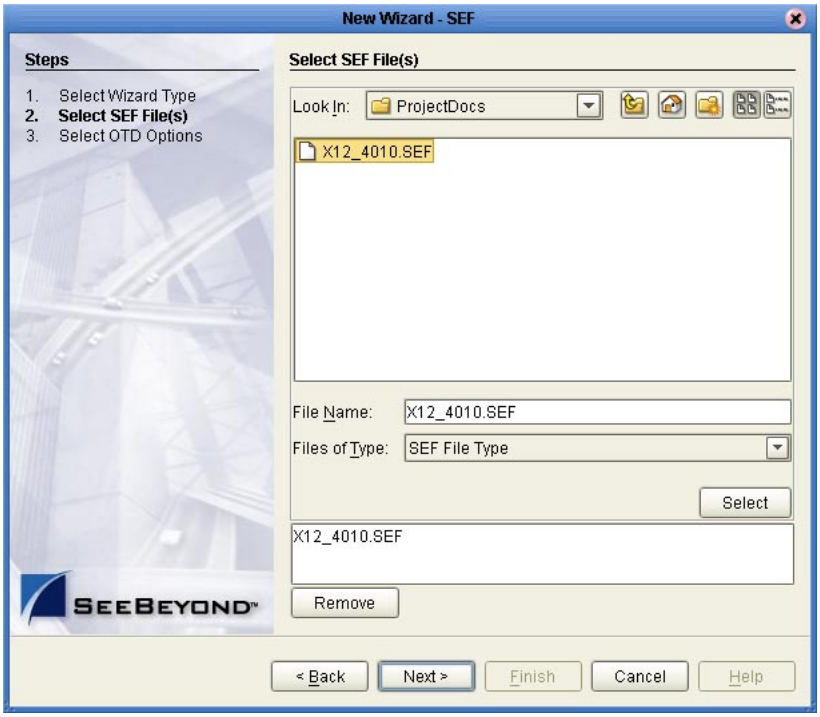
Figure 7 Creating ASC X12 OTDs



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

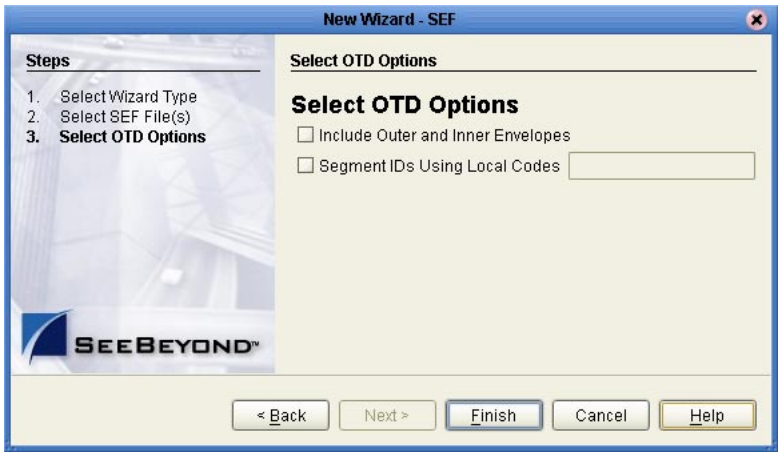
3 In the **Look In** box, navigate to the folder where the SEF file for this OTD resides, and then double-click the SEF file. This adds the file to the selection box as shown below.

Figure 8 Selecting the SEF File



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

Figure 9 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 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 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 38
- **getElementSeparator** on page 31 and **setElementSeparator** on page 39
- **getFGValidationResult** on page 32
- **getICValidationResult** on page 32
- **getInputSource** on page 32
- **getMaxDataError** on page 33 and **setMaxDataError** on page 40
- **getMaxFreedSegsComsNum** on page 33 and **setMaxFreedSegsComsNum** on page 40
- **getMaxParsedSegsComsNum** on page 33and **setMaxParsedSegsComsNum** on page 40
- **getMsgValidationResult** on page 34
- **getRepetitionSeparator** on page 34 and **setRepetitionSeparator** on page 41
- **getSegmentCount** on page 34
- **getSegmentTerminator** on page 34 and **setSegmentTerminator** on page 41
- **getSubelementSeparator** on page 35 and **setSubelementSeparator** on page 41

- **getTSValidationResult** on page 35
- **getUnmarshalError** on page 35

The following methods are available from the loop elements:

- **getLoopxxx** on page 32 and **setLoopxxx** on page 39
- **getSegmentCount** on page 34
- **setXmlOutput** on page 42

Note: *The get and set methods are automatically generated from the bean nodes. On occassion, 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.

Because the OTD automatically detects delimiters while unmarshaling, do not specify delimiters for an 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.

You can specify delimiters in two ways:

1. You can set the subelement separator and repetition separator from the corresponding elements within the ISA segment.
2. 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 38
   - **setElementSeparator** on page 39
   - **setSegmentTerminator** on page 41
   - **setSubelementSeparator** on page 41

- **setRepetitionSeparator** on page 41
- **setSubelementSeparator** on page 41)

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. If the Collaboration puts the data into ASC X12 format, you can use the set methods to set the delimiters in the output OTD. See **"Get and Set Methods" on page 27**.

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, 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 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.

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.

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 37**.

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.

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.

---

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 15**.

Exceptions

None.

marshal

Signature

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

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 12**.

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.

setxxx

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 ASC X12 delimiters:

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

For more information, refer to **"Setting Delimiters" on page 28**.

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 28**.

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 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 28.

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 28.

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 28.

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 isXML)
```

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 13**.

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 12** and **"Errors and Exceptions" on page 15**.

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 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 12** and **"performValidation" on page 37**.

What's in This Chapter

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 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>
```

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

O

P

R

S

T

U

V

X