SeeBeyond ICAN Suite

UN/EDIFACT OTD Library User's Guide

Release 5.0.2

Version 20050401065817.

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 guide contains the following information:

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

1.1.2 Scope

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

1.1.3 Intended Audience

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

1.1.4 Document Conventions

The following conventions are observed throughout this document.

Table 1 Document Conventions

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

1.1.5 Screenshots

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

1.1.6 Related Documents

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

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

- *eGate Integrator JMS Reference Guide*
- *eGate Integrator System Administrator Guide*
- *eGate Integrator Deployment Guide*
- *eXchange Integrator User's Guide*
- *eXchange Integrator Designer's Guide*
- *eInsight Business Process Manager User's Guide*
- *UN/EDIFACT Manager Composite Application User's Guide*

1.2 References

The following resources provide additional information about the UN/EDIFACT protocol:

- The United Nations Economic Commission of Europe (UN/ECE) is one of the five regional commissions of the United Nations. The UN/ECE Web site contains technical information concerning rules, standards, recent UN/EDIFACT directories, syntax, and so on.

  http://www.unece.org/trade/untdid/welcome.htm

- UN/EDIFACT publishes the messages for each version separately from the envelopes (header and trailer segments) that are used with those messages.

  The messages are published at:

  http://www.gefeg.com/en/standard/edifact/index.htm

  The envelopes are published at:

  http://www.gefeg.com/jswg/

1.3 SeeBeyond Web Site

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

http://www.seebeyond.com

1.4 SeeBeyond Documentation Feedback

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

docfeedback@seebeyond.com

Chapter 2

Overview of the UN/EDIFACT OTD Library

This chapter provides an overview of the UN/EDIFACT OTD Library as well as its support for UN/EDIFACT directory versions, SEF file versions, validation, and the UNA segment.

What's in This Chapter

2.1 About the UN/EDIFACT OTD Library

The United Nations/Electronic Data Interchange (UN/EDIFACT) for Administration, Commerce and Transport protocol was developed for the electronic exchange of machine-readable information between businesses.

The UN/EDIFACT Working Group (EWG) develops, maintains, interprets, and promotes the use of the UN/EDIFACT standard.

UN/EDIFACT messages are structured according to very strict rules. Messages are in ASCII format. The standard defines all these message elements, their sequence, and also their grouping.

UN/EDIFACT publishes the messages for each version separately from the envelopes (header and trailer segments) that are used with those messages.

The messages are available online at:

http://www.gefeg.com/en/standard/edifact/edifact.htm

The envelopes are available online at:

http://www.gefeg.com/jswg/

A new version of UN/EDIFACT messages is released several times a year, containing most of the messages in the previous version, plus any new messages that have been approved by the standards organization. The envelopes are updated with a new version infrequently.

UN/EDIFACT 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 UN/EDIFACT 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 message level 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 UN/EDIFACT OTD Library for a specific version includes message level 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 UN/EDIFACT 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 UN/EDIFACT transaction.

The list of transactions provided is different for each version of UN/EDIFACT.

The UN/EDIFACT OTD Library provides UN/EDIFACT OTDs that you can use to build ICAN Projects for interfacing with UN/EDIFACT systems. You can use the OTDs standalone with eGate Integrator or in combination with eXchange Integrator, eGate Integrator, and the UN/EDIFACT Manager Composite Application.

2.2 UN/EDIFACT Directory Support

The UN/EDIFACT OTD Library provides OTDs for the following UN/EDIFACT directories:

- D.01A and B
- D.00A and B
- D.99A and B
- D.98A and B
- D.97A and B

- D.96A and B
- D.95A and B

2.3 SEF File Support

The UN/EDIFACT 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 UN/EDIFACT OTDs from SEF Files" on page 25**.

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 UN/EDIFACT Validation Support

Within each UN/EDIFACT OTD are Java methods and Java bean nodes for handling validation (see **"performValidation" on page 40**). The marshal and unmarshal methods of the envelope OTDs handle enveloping and de-enveloping (see **"marshal" on page 39** and **"unmarshal" on page 46**). No pre-built translations are supplied with the OTD libraries; these can be built in the Java Collaboration Editor.

EDIFACT 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 31** and **"checkAll" on page 31**) is in XML format conforming to the **EDFOTDErrors.xsd** file. Refer to **"Contents of the EDFOTDErrors.xsd File" on page 47** for more information. For a sample of the validation output XML, refer to **"Sample Validation Output XML" on page 48**.

Note: *Currently the segment syntax error code (SegmSyntErroCode) and data element syntax error code (DataElemSyntErroCode) use the same codes as the X12 protocol.*

2.5 UNA Segment Support

All UN/EDIFACT messages have a UNA segment (service string advice). It is used to send delimiter and indicator characters. The UNA segment is optional per the UN/EDIFACT specification.

The string has a mandatory fixed length of 9 characters. The first three are "UNA," immediately followed by the 6 characters as defined in ISO 9735.

The UNA segment template is a fixed length with segment ID = UNA, followed by 6 one-byte fields. Each field specifies a separator or other service character. For more information, refer to **"Setting Delimiters and Indicators" on page 30**.

The OTD Library provides the *getmarshalUNA()* method to UN/EDIFACT OTD top "outer" level with its Java type of java.lang.Boolean. For information, refer to **"getMarshalUNA" on page 36**.

- If its value is java.lang.Boolean.TRUE, then UNA segment data is always included in the output message.
- If its value is java.lang.Boolean.FALSE, then UNA segment data is never included in the output message.
- If its value is null (or user never sets its value), then inclusion of UNA segment data in the output message is based on the following:

If any delimiter values are set through UNA segment object, the UNA segment data is included in the output message regardless of default or non-default delimiters are used. Otherwise,

- If non-default delimiters are used, then UNA segment data is included in the output message.
- If default delimiters are used, then UNA segment data is not included in the output message.

2.6 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, UN/EDIFACT OTDs set no limit of parsed segments or composites held in memory. You can specify a limit for parsed and freed segments or composited by using the following methods at the OTD root levels:

- setMaxParsedSegsComsNum() method (**"setMaxParsedSegsComsNum" on page 43**
- setMaxFreedSegsComsNum() method (**"setMaxFreedSegsComsNum" on page 43**)

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

2.7 Errors and Exceptions

For all UN/EDIFACT 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 14**). 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 UN/EDIFACT OTDs

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

What's in This Chapter

3.1 System Requirements

Each UN/EDIFACT OTD **.sar** file requires from 10 MB to 35 MB disk space; the combined disk space required to load all **.sar** files (v3 and v4 of D.95A through D.01B) is approximately 645 MB.

Due to the size of the UN/EDIFACT 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 18**.

Other than that, the system requirements for the UN/EDIFACT 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 UN/EDIFACT 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 UN/EDIFACT OTD Library

During the UN/EDIFACT 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 UN/EDIFACT OTDs. You must have uploaded a **license.sar** to the ICAN Repository that includes a license for the UN/EDIFACT OTD Library.

To install the UN/EDIFACT 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 **UN_EDIFACT_OTD_Lib_v3_D00A.sar** (to install version 3 of the D.00A user directory)
   - **UN_EDIFACT_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 **UN/EDIFACT OTD Library** in the left pane, and click **UN/EDIFACT 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**.

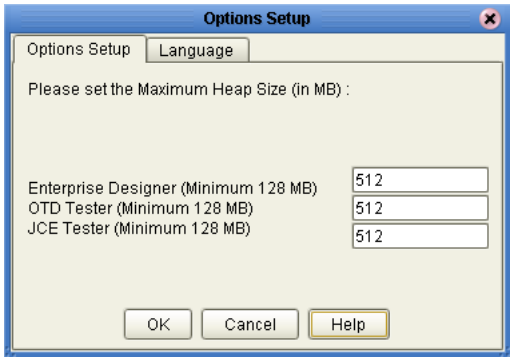
3.4 Increasing the Enterprise Designer Heap Size

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

To increase the Enterprise Designer heap size

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

Figure 1 Increasing Enterprise Designer Heap Size



3. Restart Enterprise Designer.

3.4.1 Resolving Memory Errors at Enterprise Designer Startup

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

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

Chapter 4

Using UN/EDIFACT OTDs

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

What's in This Chapter

4.1 Displaying UN/EDIFACT OTDs

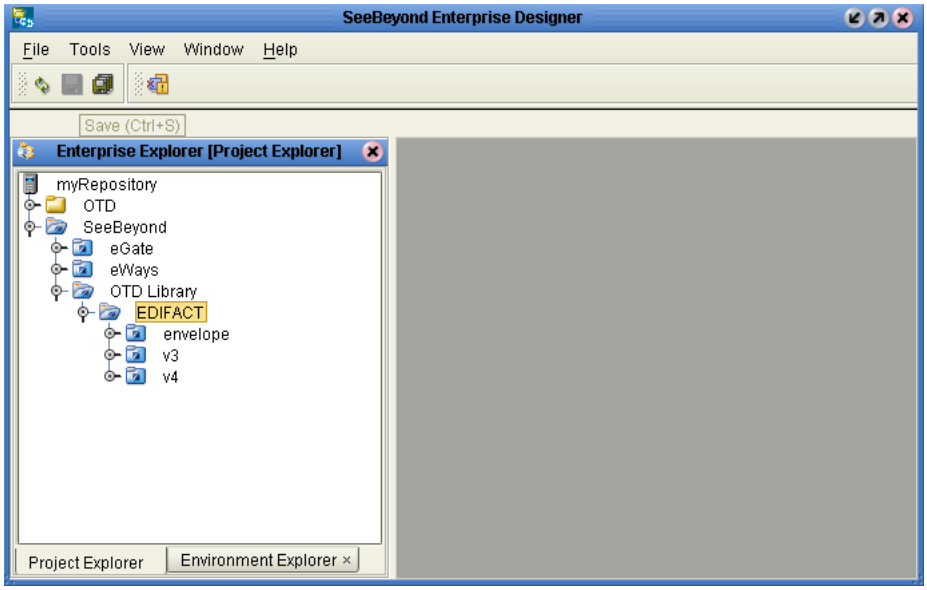
After installing the UN/EDIFACT OTDs, you can view the OTDs in the OTD Editor as described below.

To display UN/EDIFACT OTDs

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

   The Project Explorer tab displays the **Envelope**, **v3** and/or **v4** folders depending on the OTDs installed.

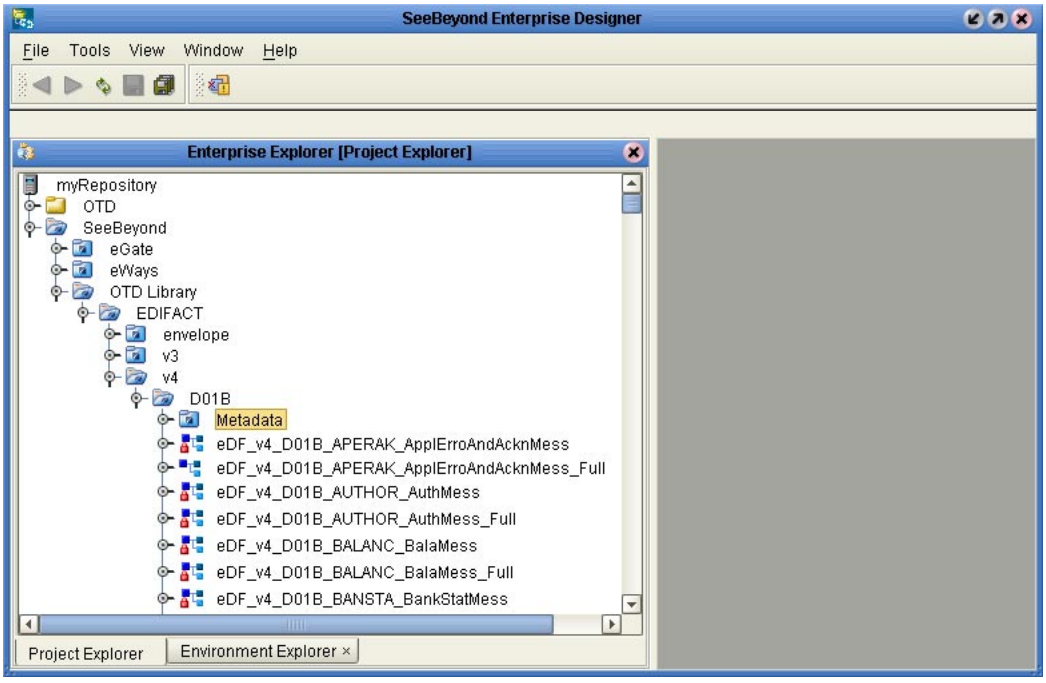
Figure 2 Finding the UN/EDIFACT OTDs in Enterprise Designer



The **v3** folder include OTDs for UN/EDIFACT version 3, and the **v4** folder includes OTDs for UN/EDIFACT version 4.

2 Expand the **v3** or **v4** folder. The folder displays the installed OTDs per UN/EDIFACT directory, for example **D01B**.

Figure 3 OTDs for UN/EDIFACT Directory D.01B Version 4



The **Project Explorer** tab displays the OTDs available for the UN/EDIFACT directory folder selected. The table below described the OTD naming conventions.

Table 2 OTD Naming Convention

eDF_	Abbreviation of the protocol name
v3_	UN/EDIFACT version 3
v4_	UN/EDIFACT version 4
D00A_	UN/EDIFACT directory
APERAK_	Abbreviation of the message name
_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 UN/EDIFACT OTDs** on page 24.

4.2 Building UN/EDIFACT OTD Collaborations

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

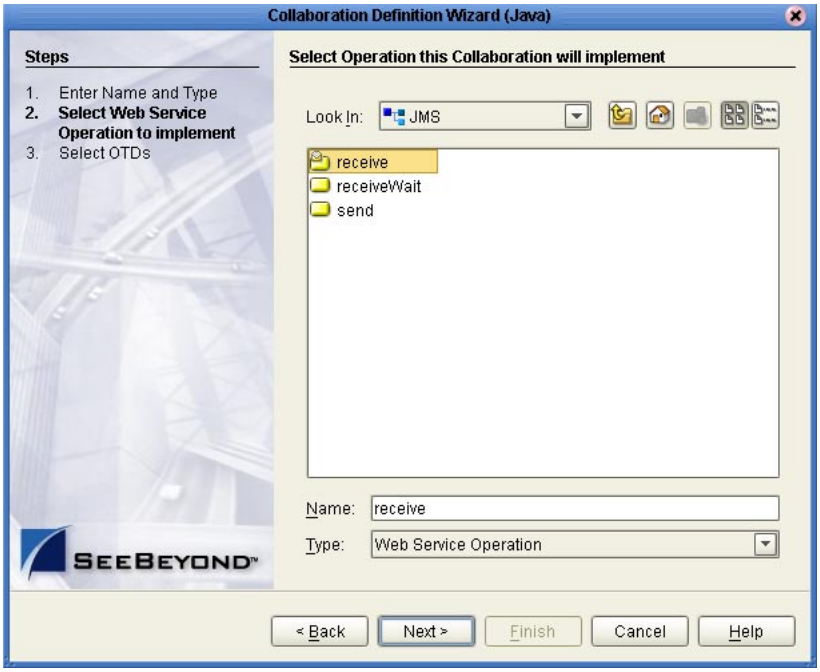
To customize the OTDs before building the Collaboration, refer to **"Customizing the UN/EDIFACT OTDs" on page 24**.

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 UN/EDIFACT OTD Library" on page 17**.

To build UN/EDIFACT 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 4 Selecting the Web Service



The **Select OTDs** page appears.

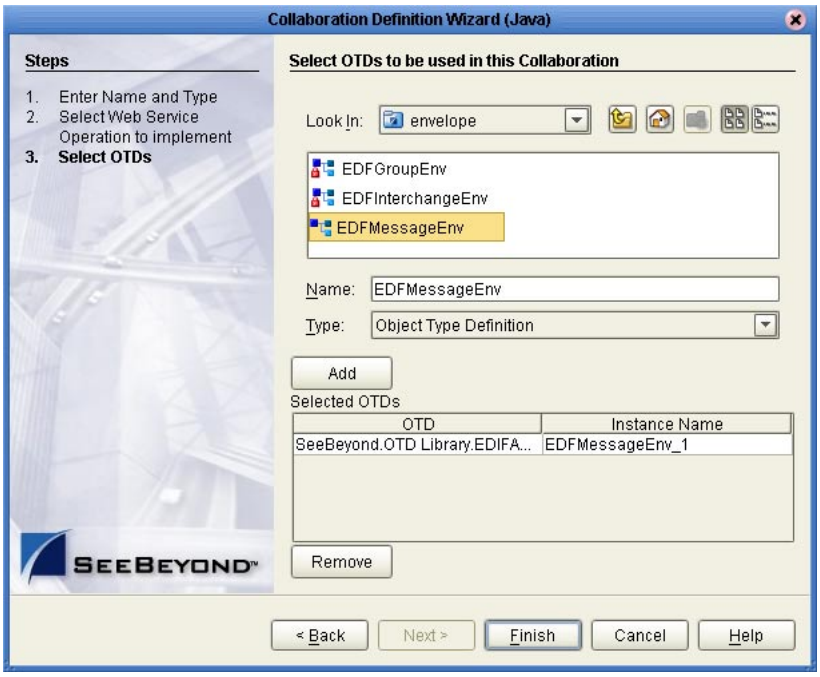
4 To use envelopes OTDs, under **Look In,** navigate to the envelopes by double-clicking the folders below. If the Collaboration does not use enveloping, continue with step 6.

- **SeeBeyond**
- **OTD Library**
- **EDIFACT**
- **Envelopes**

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 5 Adding Envelopes to the Collaboration



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

- **SeeBeyond > OTD Library > EDIFACT > v3** or **v4**
- Folder indicating the UN/EDIFACT directory, such as **D01B**

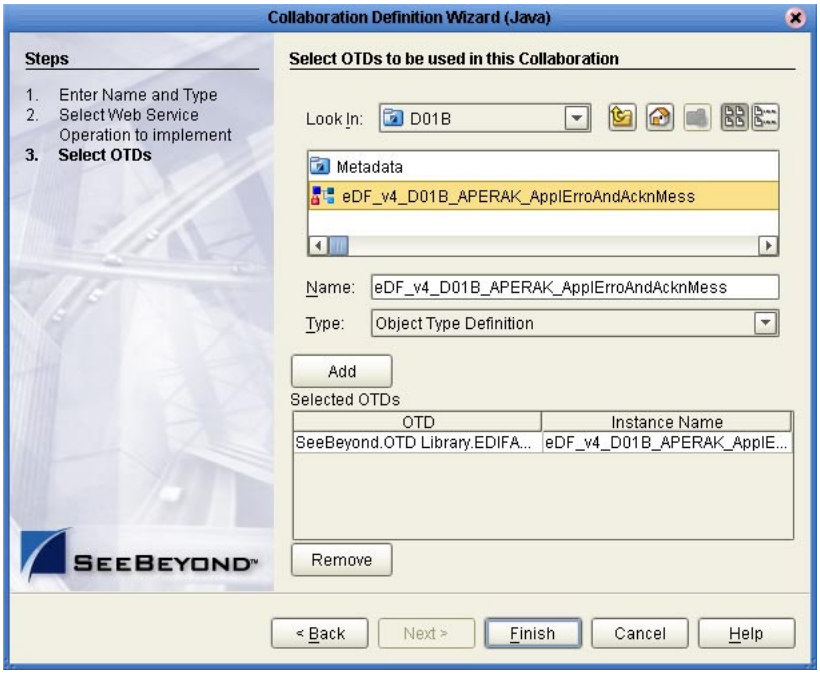
The **Look In** area displays the OTDs for the selected UN/EDIFACT directories. The table below describes the naming convention for the OTDs.

Table 3 OTD Naming Convention

eDF_	Abbreviation of the protocol name
v3_	UN/EDIFACT version 3
v4_	UN/EDIFACT version 4
D00A_	UN/EDIFACT directory
APERAK_	Abbreviation of the transaction name
_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 6 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 UN/EDIFACT OTD methods, refer to **Java Methods for UN/EDIFACT OTDs** on page 29.

4.3 Customizing the UN/EDIFACT 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.

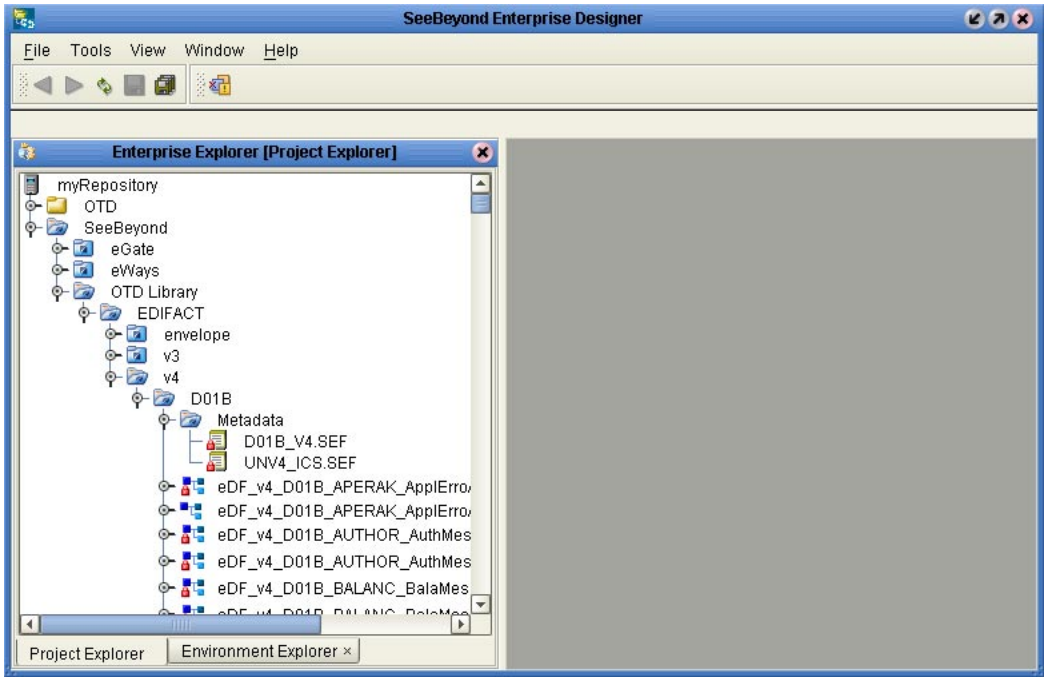
To customize UN/EDIFACT OTDs

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

- **SeeBeyond > OTD Library > EDIFACT > v3** or **v4**
- Folder indicating the UN/EDIFACT directory, such as **D01B**
- **Metadata**

The metadata folder displays the SEF files available.

Figure 7 Saving UN/EDIFACT 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 UN/EDIFACT OTDs from SEF Files

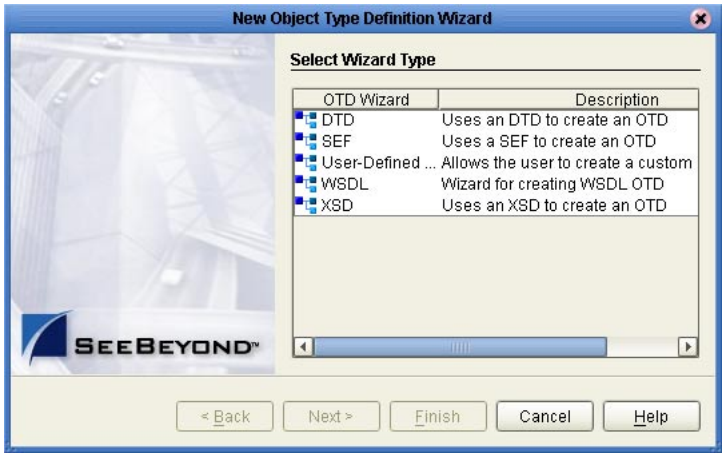
This section describes how you create UN/EDIFACT OTDs using SEF files. The UN/EDIFACT OTD Library includes the SEF files for the OTDs to allow you to customize the OTD as described in the section above. Once you have tailored the SEF file to your business requirements, you can then use the procedure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

To create OTDs from SEF files, you use the SEF OTD wizard to build the OTD using selected SEF files. The SEF OTD wizard is packaged separately from the OTD Library, so make sure that you uploaded the **SEF_OTD_Wizard.sar** to the ICAN Repository, and used the **Update Center** in Enterprise Designer to install it. For information, refer to **"Installing the UN/EDIFACT OTD Library" on page 17**.

To create UN/EDIFACT 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.

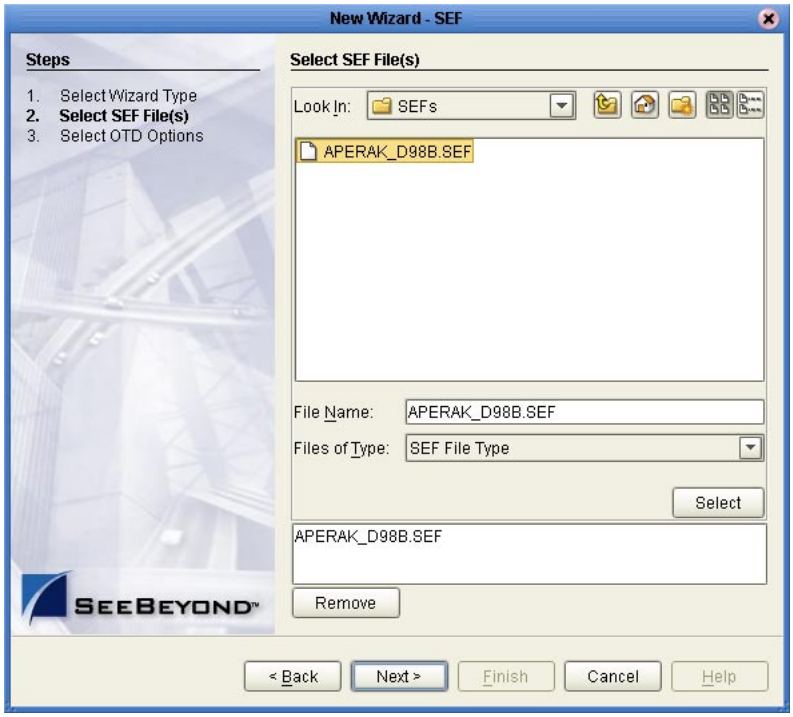
Figure 8 Creating UN/EDIFACT OTDs



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

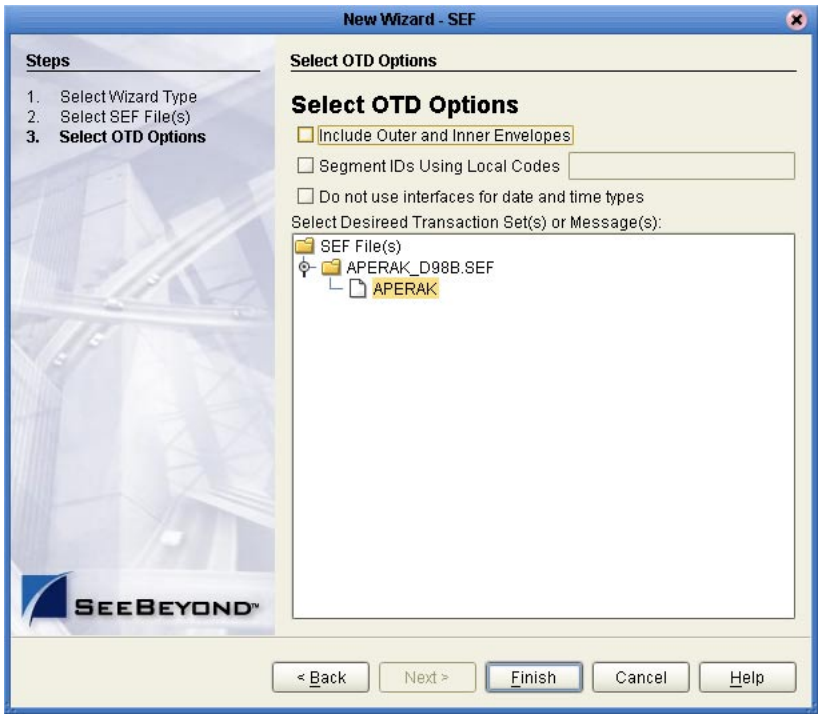
3 In the **Look In** box, navigate to the folder where the SEF file for this OTD resides, and then double-click the SEF file. This adds the file to the selection box as shown below.

Figure 9 Selecting the SEF File



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

Figure 10 Selecting the OTD Options



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

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

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

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

4.5 Possible Differences in Output When Using Pass-Through

If you are using pass-through, the output file contains essentially the same data as the input file.

Certain differences in output, based on variations in acceptable interpretation of the information, are acceptable, provided that the data conforms to the formats specified for the elements. For example:

- If the input file includes a six-digit date, the output file might represent this as an eight-digit value. For example, 040715 in the input file might be represented as 20040705 in the output file.
- The number of trailing zeros after a decimal point might vary. For example, an input value of 10.000 might be represented as 10 in the output file.

The reason these changes occur is that, during pass-through, certain data fields are parsed and stored as Java objects other than strings; for example, Date or Double.

The actual value of all the information must remain the same.

Chapter 5

Java Methods for UN/EDIFACT OTDs

This chapter describes the Java methods available for UN/EDIFACT OTDs.

What's in This Chapter

5.1 Get and Set Methods

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

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

- **getSegmentCount** on page 37
- **getSegmentTerminator** on page 37 and **setSegmentTerminator** on page 45
- **getSubelementSeparator** on page 38 and **setSubelementSeparator** on page 45
- **getTSValidationResult** on page 38
- **getUnmarshalError** on page 38

The following methods are available from the loop elements:

- **getLoopxxx** on page 34 and **setLoopxxx** on page 42
- **getSegmentCount** on page 37

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 and Indicators

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

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

When unmarshaling inbound messages, the UN/EDIFACT OTD uses delimiters specified in the UNA segment when that segment is present. If the segment is absent, the OTD uses the default industrial standard delimiters. It is unnecessary to specify delimiters for incoming messages.

For outbound messages using UN/EDIFACT OTDs, you can specify delimiters in two ways:

1. You can set the delimiter and indicator charactersfrom the corresponding elements within the UNB segment. For more information, refer to **"UNA Segment Support" on page 14**.
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 and indicators:
   - **setDecimalMark** on page 41
   - **setDefaultEdifactDelimiters** on page 41
   - **setElementSeparator** on page 42
   - **setRelease** on page 44

- **setSegmentTerminator** on page 45
- **setSubelementSeparator** on page 45
- **setRepetitionSeparator** on page 44
- **setSubelementSeparator** on page 45)

If the input data is already unmarshaled into an UN/EDIFACT OTD, you can use the get methods to retrieve the delimiters from the input data. If the Collaboration puts the data into UN/EDIFACT format, you can use the set methods to set the delimiters in the output OTD. See **"Get and Set Methods" on page 29**.

5.3 Available Methods

This section describes the signature and description for each available UN/EDIFACT OTD method.

check

Signature

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

Description

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

The method returns null if there are no validation errors.

Exceptions

None.

checkAll

Signature

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

Description

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

The method returns null if there are no validation errors.

Exceptions

None.

clone

Signature

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

Description

Creates and returns a copy of this OTD instance.

Exceptions

java.lang.CloneNotSupportedException

count*xxx*

Signature

```
public int countxxx()
```

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

Description

Counts the repetitions of the node at runtime.

Exceptions

None.

countLoop*xxx*

Signature

```
public int countLoopxxx()
```

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

Description

Counts the repetitions of the loop at runtime.

Exceptions

None.

getxxx

Signature

```
public item getxxx()
```

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

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

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

Description

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

Exceptions

None.

getAllErrors

Signature

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

Description

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

Exceptions

None.

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

```
com.stc.edifact_v3_d95B.EDF_..._...Outer myOTD=new com.stc.edifact_v3
_d95B.EDF_..._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 only at the Outer and Inner root levels of fully-evenloped OTDs.

Exceptions

None.

getICValidationResult

Signature

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

Description

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

Exceptions

None.

getMaxParsedSegsComsNum

Signature

```
public int getMaxParsedSegsComsNum()
```

Description

Returns the maximum number of segments and composite objects to be parsed.

Exceptions

None.

getMarshalUNA

Signature

```
public java.long.Boolean getMarshalUNA()
```

Description

Returns the Boolean value to indicate whether or not the UNA segment is to be marshaled. This method is only available at the top "outer" level of the OTD.

- if the return value is java.lang.Boolean.TRUE, then UNA segment data is always included in the output message.
- if the return value is java.lang.Boolean.FALSE, then UNA segment data is never included in the output message.
- if the return value is null (or user never sets its value), then inclusion of UNA segment data in the output message is based on the following:

If any delimiter values are set through UNA segment object, the UNA segment data is included in the output message regardless of default or non-default delimiters are used. Otherwise,

- if non-default delimiters are used, then UNA segment data is included in the output message.
- if default delimiters are used, then UNA segment data is not included in the output message.

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 40**.

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

Exceptions

None.

getRelease

Signature

```
public char getRelease()
```

Description

Returns the release character.

Exceptions

None.

getRepetitionSeparator

Signature

```
public char getRepetitionSeparator()
```

Description

Returns the repetition separator character.

Exceptions

None.

Examples

```
com.stc.edifact_v3_d95B.EDF_..._...Outer myOTD=new com.stc.edifact_v3
_d95B.EDF_..._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

```
com.stc.edifact_v3_d95B.EDF_..._...Outer myOTD=new com.stc.edifact_v3
_d95B.EDF_..._Outer();
......
......
char segTerm=myOTD.getSegmentTerminator();
```

getSubelementSeparator

Signature

```
public char getSubelementSeparator()
```

Description

Returns the subelement/composite element separator character.

Exceptions

None.

Example

```
com.stc.edifact_v3_d95B.EDF_..._...Outer myOTD=new com.stc.edifact_v3
_d95B.EDF_..._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 method is available only in the Outer, Inner, and message root levels of 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.

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

Exceptions

None.

marshal

Signature

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

Description

Marshals the internal data tree into an output stream. For more information, see **"On Demand Parsing" on page 14** .

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 **"UN/EDIFACT Validation Support" on page 13**.

Exceptions

None.

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.

setDecimalMark

Signature

```
public void setDecimalMark(char)
```

Description

Sets the decimal mark.

Exceptions

None.

setDefaultEdifactDelimiters

Signature

```
public void setDefaultEdifactDelimiters()
```

Description

Sets the current delimiters to the default UN/EDIFACT delimiters:

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

For more information, refer to **"Setting Delimiters and Indicators" on page 30**.

Exceptions

None

Example

```
com.stc.edifact_v3_d95B.EDF_..._...Outer myOTD=new com.stc.edifact_v3
_d95B.EDF_..._Outer();
......
......
myOTD.setDefaultEdifactDelimiters();
```

setElementSeparator

Signature

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

Description

Sets the element separator character. For more information, refer to **"Setting Delimiters and Indicators" on page 30**.

Exceptions

None

Examples

```
com.stc.edifact_v3_d95B.EDF_..._...Outer myOTD=new com.stc.edifact_v3
_d95B.EDF_..._Outer();
......
......
char c='+';
myOTD.setElementSeparator(c);
```

setLoopxxx

Signature

```
public void set<i>Loopxxx</i>(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<i>Loopxxx</i>(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 14**.

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 14**.

Exceptions

None.

setMarshalUNA

Signature

```
public void setMarshalUNA (java.long.Boolean)
```

Description

Sets the Boolean value to indicate whether or not the UNA segment is to be marshaled. This method is only available at the top "outer" level of the OTD.

- If the *item* is java.lang.Boolean.TRUE, then UNA segment data is always included in the output message.
- If the *item* is java.lang.Boolean.FALSE, then UNA segment data is never included in the output message.
- If the *item* is null (or user never sets its value), then inclusion of UNA segment data in the output message is based on the following:

If any delimiter values are set through UNA segment object, the UNA segment data is included in the output message regardless of default or non-default delimiters are used. Otherwise,

- if non-default delimiters are used, then UNA segment data is included in the output message.
- if default delimiters are used, then UNA segment data is not included in the output message.

For more information, refer to **"UNA Segment Support" on page 14**.

Exceptions

None.

setRelease

Signature

```
public void setRelease(char)
```

Description

Sets the release character.

Exceptions

None.

setRepetitionSeparator

Signature

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

Description

Sets the repetition separator character. For more information, refer to **"Setting Delimiters and Indicators" on page 30**.

Exceptions

None.

Example

```
com.stc.edifact_v3_d95B.EDF_..._...Outer myOTD=new com.stc.edifact_v3
_d95B.EDF_..._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 and Indicators" on page 30**.

Exceptions

None.

Example

```
com.stc.edifact_v3_d95B.EDF_..._...Outer myOTD=new com.stc.edifact_v3
_d95B.EDF_..._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 and Indicators" on page 30**.

Exceptions

None.

Example

```
com.stc.edifact_v3_d95B.EDF_..._...Outer myOTD=new com.stc.edifact_v3
_d95B.EDF_..._Outer();
......
```

```
......
char c=':';
myOTD.setSubelementSeparator(c);
```

unmarshal

Signature

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

Description

Unmarshals the given input into an internal data tree.

For more information, refer to **"On Demand Parsing" on page 14** 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 (deserializes, parses) the given input string into an internal data tree.

Exceptions

java.io.IOException for input problems

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

Appendix A

EDFOTDErrors Schema File and Sample XML

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

For more information, refer to **"UN/EDIFACT Validation Support" on page 13** and **"performValidation" on page 40**.

What's in This Chapter

6.1 Contents of the EDFOTDErrors.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="EDFOTDErrors">
    <xs:annotation>
      <xs:documentation>Validation Errors from an EDF OTD validation</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="EDFICError" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="EDFFGError" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="EDFTSError" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="EDFDataError" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="EDFICError">
    <xs:annotation>
      <xs:documentation>Interchange Envelope Validation Error Structure.</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="EDFFGError">
    <xs:annotation>
      <xs:documentation>Functional Group Envelope Validation Error Structure.</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="EDFTSError">
    <xs:annotation>
      <xs:documentation>Transaction Set Envelope Validation Error Structure.</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="EDFDataError">
    <xs:annotation>
      <xs:documentation>Message (excluding envelopes) Validation Error Structure.</
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 Validation Output XML

```
<EDFOTDErrors>
  <EDFDataError>
    <Level>1</Level>
    <SegmIDCode>QTY</SegmIDCode>
    <SegmPosiInTranSet>24</SegmPosiInTranSet>
    <LoopIdenCode>QTY</LoopIdenCode>
    <SegmSyntErroCode>8</SegmSyntErroCode>
    <ElemPosiInSegm>2</ElemPosiInSegm>
    <DataElemSyntErroCode>3</DataElemSyntErroCode>
    <CopyOfBadDataElem>50:PCE</CopyOfBadDataElem>
    <ErrorCode>15037</ErrorCode>
    <ErrorDesc>QTY_QTY_2 at 24 [50:PCE]: Number of data elements inside the segment during
unmarshalling exceeds 1</ErrorDesc>
    <Severity>ERROR</Severity>
  </EDFDataError>
  <EDFDataError>
    <Level>1</Level>
    <SegmIDCode>QTY</SegmIDCode>
    <SegmPosiInTranSet>26</SegmPosiInTranSet>
    <LoopIdenCode>QTY</LoopIdenCode>
    <SegmSyntErroCode>8</SegmSyntErroCode>
    <ElemPosiInSegm>1</ElemPosiInSegm>
    <CompDataElemPosiInComp>2</CompDataElemPosiInComp>
    <DataElemRefeNumb>6060</DataElemRefeNumb>
    <DataElemSyntErroCode>1</DataElemSyntErroCode>
    <ErrorCode>15040</ErrorCode>
    <ErrorDesc>QTY_QTY_1 at 26: Data subelement is required but missing inside the composite during
unmarshalling</ErrorDesc>
    <Severity>ERROR</Severity>
  </EDFDataError>
  <EDFDataError>
    <Level>1</Level>
    <SegmIDCode>DTM</SegmIDCode>
    <SegmPosiInTranSet>5</SegmPosiInTranSet>
    <LoopIdenCode>RFF</LoopIdenCode>
    <SegmSyntErroCode>8</SegmSyntErroCode>
    <ElemPosiInSegm>1</ElemPosiInSegm>
    <CompDataElemPosiInComp>1</CompDataElemPosiInComp>
    <DataElemRefeNumb>2005</DataElemRefeNumb>
    <DataElemSyntErroCode>7</DataElemSyntErroCode>
    <CopyOfBadDataElem>004</CopyOfBadDataElem>
```

```
    <ErrorCode>15063</ErrorCode>
    <ErrorDesc>RFF_DTM_1 at 5 [004]: Code value is not in the code list of
2,3,4,7,8,9,10,11,12,13,14,15,16,17,18,20,21,22,35,36</ErrorDesc>
    <Severity>ERROR</Severity>
  </EDFDataError>
  <EDFDataError>
    <Level>1</Level>
    <SegmIDCode>NAD</SegmIDCode>
    <SegmPosiInTranSet>7</SegmPosiInTranSet>
    <LoopIdenCode>NAD</LoopIdenCode>
    <SegmSyntErroCode>8</SegmSyntErroCode>
    <ElemPosiInSegm>4</ElemPosiInSegm>
    <CompDataElemPosiInComp>1</CompDataElemPosiInComp>
    <DataElemRefeNumb>3036</DataElemRefeNumb>
    <DataElemSyntErroCode>5</DataElemSyntErroCode>
    <CopyOfBadDataElem>VOLVO AERO CORPORATION S-461 81 TROLLHATTAN</CopyOfBadDataElem>
    <ErrorCode>15055</ErrorCode>
    <ErrorDesc>NAD_NAD_4 at 7 [VOLVO AERO CORPORATION S-461 81 TROLLHATTAN]: Data has too many
characters of 43 because less_or_equal 35</ErrorDesc>
    <Severity>ERROR</Severity>
  </EDFDataError>
  <EDFDataError>
    <Level>1</Level>
    <SegmIDCode>PAT</SegmIDCode>
    <SegmPosiInTranSet>12</SegmPosiInTranSet>
    <LoopIdenCode>PAT</LoopIdenCode>
    <SegmSyntErroCode>8</SegmSyntErroCode>
    <ElemPosiInSegm>2</ElemPosiInSegm>
    <CompDataElemPosiInComp>1</CompDataElemPosiInComp>
    <DataElemRefeNumb>4277</DataElemRefeNumb>
    <DataElemSyntErroCode>7</DataElemSyntErroCode>
    <CopyOfBadDataElem>30</CopyOfBadDataElem>
    <ErrorCode>15063</ErrorCode>
    <ErrorDesc>PAT_PAT_2 at 12 [30]: Code value is not in the code list of 1,2,3,4,5,6</ErrorDesc>
    <Severity>ERROR</Severity>
  </EDFDataError>
  <EDFDataError>
    <Level>1</Level>
    <SegmIDCode>QTY</SegmIDCode>
    <SegmPosiInTranSet>24</SegmPosiInTranSet>
    <LoopIdenCode>QTY</LoopIdenCode>
    <SegmSyntErroCode>8</SegmSyntErroCode>
    <ElemPosiInSegm>1</ElemPosiInSegm>
    <CompDataElemPosiInComp>2</CompDataElemPosiInComp>
    <DataElemRefeNumb>6060</DataElemRefeNumb>
    <DataElemSyntErroCode>4</DataElemSyntErroCode>
    <CopyOfBadDataElem/>
    <ErrorCode>15056</ErrorCode>
    <ErrorDesc>QTY_QTY_1 at 24 []: Data has too few characters of 0 because greater_or_equal 1</
ErrorDesc>
    <Severity>ERROR</Severity>
  </EDFDataError>
</EDFOTDErrors>
```

Index

O

P

R

S

T

U

V