Oracle® Java CAPS Message Library for EDIFACT User's Guide

Part No: 821–2607
March 2011

ORACLE®

120126@25097

Contents

Using the Message Library for EDIFACT

This document provides information and instructions for using the Java CAPS EDIFACT Message Library.

What You Need to Know

The following topics provide information you need to know when working with the EDIFACT Message Library.

- "Overview of the EDIFACT Message Library" on page 5
- "Java Methods for UN/EDIFACT OTDs" on page 15
- "EDFOTDErrors Schema File and Sample XML" on page 34

What You Need to Do

The following topics provide instructions on how to use the Java CAPS EDIFACT Message Library.

- "Installing the Message Library for EDIFACT" on page 10
- "Using UN/EDIFACT Message Libraries" on page 10

Overview of the EDIFACT Message Library

This topic provides an overview of the EDIFACT Message Library as well as its support for EDIFACT directory versions, SEF file versions, validation, and the UNA segment. It includes the following topics:

- "About the EDIFACT Message Library" on page 6
- "References" on page 7
- "Character Encoding" on page 7
- "UN/EDIFACT Directory Support" on page 7
- "SEF File Support" on page 8
- "UN/EDIFACT Validation Support" on page 8

About the EDIFACT Message 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/`.

New versions of UN/EDIFACT messages are 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 specific message structure that indicates how data elements are organized and related to each other for a particular EDI transaction. In Java CAPS, the message structures are defined in an Object Type Definition (OTD), which 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.

The Oracle Java CAPS Enterprise Service uses message libraries 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 and the list of transactions provided is different for each version of UN/EDIFACT.

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/edifact.htm

  The envelopes are published at:

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

Character Encoding

The EDIFACT OTDs expect the input for unmarshalling to be a String/text when using the `unmarshalFromString` method, or to be a UTF-8 encoded byte array when using the `unmarshalFromBytes` method. If you use the File Adapter to read the content of a file as a byte array, make sure that the byte array is UTF-8 encoded by explicitly setting the encoding to UTF-8. For example, the following returns a UTF-8 encoded byte array for the File Adapter client named fileClient:

```
byte[] content = fileClient.getText().getBytes( "UTF-8" );
```

UN/EDIFACT Directory Support

The EDIFACT Message Library provides message structures for the following UN/EDIFACT directories, both syntax versions 3 and 4:

- D.03A
- 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

SEF File Support

The EDIFACT Message Library support SEF versions 1.5 and 1.6 when the SEF OTD wizard is used to build custom object type definitions (OTDs). 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.

UN/EDIFACT Validation Support

Within each EDIFACT OTD are Java methods and Java bean nodes for handling validation (see "performValidation" on page 27). The marshal and unmarshal methods of the envelope structures handle enveloping and de-enveloping (see "marshal" on page 26 and "unmarshal" on page 33). No pre-built translations are supplied with the message 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 17 and "checkAll" on page 17) is in XML format conforming to the `EDFOTDErrors.xsd` file. Refer to "Contents of the EDFOTDErrors.xsd File" on page 35 for more information. For a sample of the validation output XML, refer to "Sample Validation Output XML" on page 36.

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

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 of 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 16.

- If any delimiter values are set through UNA segment object, UNA segment data is included in the output message regardless of default or non-default delimiters are used.
- If non-default delimiters are used and no values are set through UNA segment object, UNA segment data is included in the output message.
- If default delimiters are used and no values are set through UNA segment object, UNA segment data is not included in the output message.

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 message library 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 `getxxx()` method. The OTD may assign unmarshaled segments and composites to a pool that is ready to be freed from memory by the Java Virtual Machine (JVM). Once these segments or composites are freed from memory, they become unparsed. If the element within segment or composite is accessed again, the message library reparses the segment or composite.

By default, UN/EDIFACT message libraries set no limit of parsed segments or composites held in memory. You can specify a limit for parsed and freed segments or composite using the following methods at the message library root levels:

- "setMaxParsedSegsComsNum" on page 31 method
- "setMaxFreedSegsComsNum" on page 31 method

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

Errors and Exceptions

For all UN/EDIFACT message libraries, including the envelope libraries, if the incoming message cannot be parsed (for example, if the library 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()` ran without reporting any errors. Successful completion does not guarantee that the message library instance is free of unmarshal exceptions within segments, however, since elements are

not unmarshaled until the first `getElementXxxx()` method of a segment is encountered (see "On Demand Parsing" on page 9). Encountering the unmarshal exception 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.

Installing the Message Library for EDIFACT

The UN/EDIFACT Message Library is not installed as part of the standard Java CAPS installation. You can install the library components you need using the Java CAPS Suite installer. Instructions for installing the UN/EDIFACT Message Library to both the Repository and the NetBeans IDE are provided in "Installing Additional Repository-Based Java CAPS Components" in *Installing Additional Components for Oracle Java CAPS 6.3*.

The `.sar` files for installing the UN/EDIFACT Message Library are located on your installation media in the `components/message_libraries/EDIFACT` folder. The file names include the version and the syntax version numbers so you know which ones to install.

Each UN/EDIFACT Message Library `.sar` file requires from 10 to 35 MB of disk space; the combined disk space required to load all `.sar` files is approximately 750 MB.

Using UN/EDIFACT Message Libraries

This topic includes instructions for working with the UN/EDIFACT message structures, or OTDs, provided in the UN/EDIMessagessage Library, including customizing message structures and building UN/EDIFACT Collaborations.

The following topics provide instructions for working with the message library:

- "Displaying UN/EDIFACT OTDs" on page 10
- "Building UN/EDIFACT OTD Collaborations" on page 11
- "Customizing the UN/EDIFACT OTDs" on page 13
- "Creating UN/EDIFACT OTDs from SEF Files" on page 13
- "Possible Differences in Output When Using Pass-Through" on page 14

Displaying UN/EDIFACT OTDs

After installing the UN/EDIFACT Message Library, you can view the message structures in the OTD Editor as described below.

To Display UN/EDIFACT OTDs

1. **Make sure you are connected to the Java CAPS Repository from NetBeans.**

2. **In the Projects window, expand CAPS Components Library, expand Message Library, and then expand EDIFACT.**

   Depending on which message libraries you installed, the envelope, v3, or v4 folders are visible. The v3 folder includes files for UN/EDIFACT version 3, and the v4 folder includes files for UN/EDIFACT version 4.

3. **Expand the v3 or v4 folder.**

   Nodes for each installed UN/EDIFACT directory appear, such as D01B. The table below describes the naming conventions for the OTDs.

eDF_	Abbreviation of the protocol name
v3_ or **v4_**	Indicates the UN/EDIFACT version
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 library. You can use the SEF files to customize the OTD as described in "Customizing the UN/EDIFACT OTDs" on page 13.

Building UN/EDIFACT OTD Collaborations

This section describes how to build Java Collaborations that use the UN/EDIFACT OTDs provided in the UN/EDIFACT Message Library. To customize the OTDs before building the Collaboration, refer to "Customizing the UN/EDIFACT OTDs" on page 13.

To Build UN/EDIFACT OTD Collaborations

1. **In the NetBeans Projects window, right-click the Project for which you want to create a Collaboration, point to New, and select Collaboration Definition (Java).**

   The Collaboration Definition Wizard dialog box appears.

2. **Enter the name of the Collaboration and specify whether to use and existing or new web service operation.**

3 **Click Next.**

The next page to appear depends on the web service type you selected.

4 **Do one of the following:**

- **If you selected an existing web service, do the following:**
- **If you selected a new web service, enter a name for the operation and then click Next.**

  a. **Browse to and select the web service to use, and then click Next.**

  b. **Browse to and select the input message to use, and then click Next.**

  c. **Browse to and select the output message to use, and then click Next.**

5 **To use envelope OTDs, navigate to envelopes in the Look In field and double-click the envelopes to use.**

Envelopes are located in `\CAPS Component Library\Message Library\EDIFACT\envelopes`.

The envelope names appear in the Selected OTDs section of the wizard.

6 **To select message OTDs, navigate to the UN/EDIFACT directory in the Look In field.**

The UN/EDIFACT directories are located in `\CAPS Component Library\Message Library\EDIFACT\`*version*, where *version* is either v3 or v4.

The Look In field displays the OTDs for the selected UN/EDIFACT directory. The table below describes the naming convention for the OTDs.

eDF_	Abbreviation of the protocol name
v3_ or **v4_**	UN/EDIFACT version
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 use.**

The OTDs appear under Selected OTDs.

8 **Click Finish.**

The Collaboration appears in the Collaboration Editor. You can now use the Java CAPS and OTD methods to build the business logic for the Collaboration. For information about the UN/EDIFACT OTD methods, see "Java Methods for UN/EDIFACT OTDs" on page 15.

Customizing the UN/EDIFACT OTDs

The OTDs provided in the OTD Library cannot be customized. However, the Message Library provides the SEF files so you can 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 NetBeans Projects window, expand CAPS Component Library, expand Message Library, expand EDIFACT, and then expand v3 or v4.**
2. **Expand the folder for the UN/EDIFACT directory you want to work with, such as D01B, and then expand Metadata.**

   The metadata folder displays the available SEF files.
3. **Right-click the SEF file you want to customize, and then click Export.**

   The Save As dialog box appears.
4. **Select a location for the SEF file and click Save.**
5. **Use a SEF editor to customize the file.**
6. **Use the SEF OTD wizard to rebuild the OTD as described in the next section.**

Creating UN/EDIFACT OTDs from SEF Files

This section describes how to create UN/EDIFACT OTDs using SEF files. The UN/EDIFACT Message Library includes the SEF files for the OTDs so you can customize the OTD. Once you have tailored the SEF file to your business requirements, you can 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 Message Library, so make sure you have installed it. For information, see "Installing the Message Library for EDIFACT" on page 10.

▼ To Create UN/EDIFACT OTDs from SEF files

1. **In the NetBeans Projects window, right-click the Project, point to New, and then click Object Type Definition.**

   The New Object Type Definition dialog box appears.

2. **Select SEF in the OTD Wizard list, and then Next.**

   The Select SEF File(s) page appears.

3. **In the Look In field, navigate to the folder where the SEF file for this OTD resides, and then double-click the SEF file.**

4. **Click Select, and when the file appears in the box at the bottom of the page, click Next.**

   The Select OTD Options page appears.

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. **Select the transaction sets you want to include using the Ctrl and Shift keys to select multiple options.**

9. **Click Finish.**

   The OTD Editor appears, displaying the OTD.

Possible Differences in Output When Using Pass-Through

If you are using a pass-through Collaboration, 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.

Java Methods for UN/EDIFACT OTDs

This topic describes the Java methods available for UN/EDIFACT OTDs and contains the following topics:

Note – You can view complete method information for the EDIFACT OTDs by opening the Class Browser dialog box from the Java Collaboration Definition Editor toolbar.

Get and Set Methods

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

The following get and set methods are available under the root node and at the xxx_Outer, xxx_Inner, and xxx levels:

- "getDecimalMark" on page 19 and "setDecimalMark" on page 29
- "setDefaultEDFDelimiters" on page 29
- "getElementSeparator" on page 19 and "setElementSeparator" on page 29
- "getFGValidationResult" on page 20
- "getICValidationResult" on page 20
- "getInputSource" on page 21
- "getMaxDataError" on page 21 and "setMaxDataError" on page 30
- "getMaxFreedSegsComsNum" on page 22 and "setMaxFreedSegsComsNum" on page 31
- "getMaxParsedSegsComsNum" on page 22and "setMaxParsedSegsComsNum" on page 31
- "getMsgValidationResult" on page 22
- "getRelease" on page 23 and "setRelease" on page 31
- "getRepetitionSeparator" on page 23 and "setRepetitionSeparator" on page 32
- "getSegmentCount" on page 23
- "getSegmentTerminator" on page 24 and "setSegmentTerminator" on page 32
- "getSubelementSeparator" on page 24 and "setSubelementSeparator" on page 33
- "getTSValidationResult" on page 25
- "getUnmarshalErrors" on page 25

The following methods are available from the loop elements:

- "getLoopxxx" on page 21 and "setLoopxxx" on page 30
- "getSegmentCount" on page 23

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

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 OTDs use 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 characters from the corresponding elements within the UNB segment. For more information, refer to "UNA Segment Support" on page 8.
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 29
   - "setDefaultEDFDelimiters" on page 29
   - "setElementSeparator" on page 29
   - "setRelease" on page 31
   - "setSegmentTerminator" on page 32
   - "setSubelementSeparator" on page 33
   - "setRepetitionSeparator" on page 32

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 15.

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-enveloped OTDs.

Exceptions

None.

getICValidationResult

Signature

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

Description

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

Exceptions

None.

getInputSource

Signature

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

Description

Returns the byte array of the original input data source.

Exceptions

None.

getLoopxxx

Signature

`public` *item* `get`*Loopxxx*`()`

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

`public` *item[]* `get`*Loopxxx*`()`

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.

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 27.

This 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.

getUnmarshalErrors

Signature

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

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 has*xxx*()

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 9 and "Errors and Exceptions" on page 9.

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 9 .

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, see "UN/EDIFACT Validation Support" on page 8.

Exceptions

None.

reset

Signature

```
public void reset()
```

Description

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

Exceptions

None.

set*xxx*

Signature

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

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

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

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

Description

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

Exceptions

None.

setDecimalMark

Signature

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

Description

Sets the decimal mark.

Exceptions

None.

setDefaultEDFDelimiters

Signature

```
public void setDefaultEDFDelimiters()
```

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 16.

Exceptions

None

Example

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

setElementSeparator

Signature

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

Description

Sets the element separator character. For more information, refer to "Setting Delimiters and Indicators" on page 16.

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`*`Loopxxx`*`(`*`item`*`)`

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

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

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

Description

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

Exceptions

None.

setMaxDataError

Signature

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

Description

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

Exceptions

None.

setMaxFreedSegsComsNum

Signature

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

Description

Sets the maximum number of segment and composite objects marked to be freed from memory. For more information, see "On Demand Parsing" on page 9.

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 9.

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 16.

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, see "Setting Delimiters and Indicators" on page 16.

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, see "Setting Delimiters and Indicators" on page 16.

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, see "On Demand Parsing" on page 9 and "Errors and Exceptions" on page 9.

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.

EDFOTDErrors Schema File and Sample XML

This section provides the contents of the `EDFOTDErrors.xsd` file, which is the schema file for the validation output string. This section also includes a sample of validation XML output. For more information, see "UN/EDIFACT Validation Support" on page 8 and "performValidation" on page 27.

This section contains the following topics:

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>
```

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>
```