IVI Technologies

Integrity Value Innovation

Stylus XML Converters® For Java

User's Guide

Third Party Acknowledgments: There are no products in the Stylus XML Converters release that include third party component covered by licenses that require documentation notices be provided.

Table of Contents

Stylus XML Converters User's Guide for Java

This help is your guide and reference to the Stylus XML Converters and describes how to use them to build Java applications that provide bi-directional access to non-XML data.

This help provides information about the following topics:

- The converter: URI scheme
- Using Stylus XML Converters to convert non-XML sources (such as EDI and legacy file formats) to XML
- Using Stylus XML Converters to convert XML to non-XML format (such as CSV and tab-delimited files)
- Examples and tutorials that show how you can use Stylus XML Converters in your environment
- Stylus XML Converters properties reference

What are Stylus XML Converters?

Stylus XML Converters are high-performance Java components that provide bi-directional, programmatic access to virtually any non-XML file including EDI, flat files, and other legacy formats. Stylus XML Converters allow developers to seamlessly stream non-XML data as XML to industry-leading XML processing components or to an application. They support StAX, SAX, XMLReader, XMLWriter, DOM, and Input/Output streaming interfaces. They can be embedded directly for translation purposes or as part of a chain of programs including XSLT and XQuery, or even inside XML pipelines.

Stylus XML Converters maximize developer productivity and provide a fast, scalable solution for converting between EDI and other legacy formats and XML.

Using This Help File

This help describes Stylus XML Converters and provides information on how to use them to develop Java applications. It is assumed that you are familiar with XML, Java, and related technologies.

This book help covers the following topics:

- Chapter , “Overview” provides an overview of the Stylus XML Converters API and URI schemes used for data integration.
- Chapter 2, “Stylus XML Converters® URI Schemes” describes the converter: URI scheme and how to use Stylus Studio® XML Enterprise Suite to build converter: URIs.
- Chapter 3, “Analyzing EDI-to-XML Conversions” describes how to use the Stylus XML Converters API to analyze EDI streams for errors, generate an analysis report in XML format, and manage transmission response messages as part of the conversion process.
- Chapter 4, “Stylus XML Converters® Examples” describes demo.java, a simple Java program installed with Stylus XML Converters, including how to run it, and detailed information about the actions performed by the example applications it contains. Other uses of the Java API are also illustrated.
- Chapter 5, “Stylus XML Converters® Properties” describes values for the properties for Stylus XML Converters.

For the latest information about Stylus XML Converters, see the README file in your software package or refer to the Web site:

https://www.xmlconverters.com

NOTE: This help refers the reader to Web URLs for more information about specific topics, including Web URLs not maintained by IVI Technologies. Because it is the nature of Web content to change frequently, IVI Technologies can guarantee that the URLs referenced in this book were correct at the time of publishing.

Typographical Conventions

This help file uses the following typographical conventions:

Convention	Explanation
italics	Introduces new terms that you may not be familiar with, and is used occasionally for emphasis.
bold	Emphasizes important information. Also indicates button, menu, and icon names on which you can act. For example, click **Next**.
UPPERCASE	Indicates keys or key combinations that you can use. For example, press the ENTER key.
`monospace`	Indicates syntax examples, values that you specify, or results that you receive.
`monospaced italics`	Indicates names that are placeholders for values you specify; for example, *`filename`*.
forward slash /	Separates menus and their associated commands. For example, Select File / Copy means to select Copy from the File menu.
vertical rule |	Indicates an OR separator to delineate items.
brackets [ ]	Indicates optional items. For example, in the following statement: SELECT [DISTINCT], DISTINCT is an optional keyword.
braces { }	Indicates that you must select one item. For example, {yes | no} means you must specify either yes or no.

Convention	Explanation
ellipsis . . .	Indicates that the immediately preceding item can be repeated any number of times in succession. An ellipsis following a closing bracket indicates that all information in that unit can be repeated.

About the Product Documentation

The Stylus XML Converters library consists of the following books:

- *Stylus XML Converters® for Java™ Installation Guide* describes the requirements and procedures for installing Stylus XML Converters.
- *Stylus XML Converters® for Java™ User's Guide* provides information about using Stylus XML Converters to write applications that provide bi-directional, programmatic access to non-XML files including EDI, flat files, and other legacy formats.

HTML Version

An HTML version of theStylus XML Converters documentation is placed on your system during a normal installation of the product. This documentation is located in the *installdir*\help directory. To use the help, you must have one of the following browsers installed:

- Internet Explorer 7.*x* or higher
- Mozilla Firefox 8.*x* or higher
- Safari 4.*x* or higher
- Opera 8.*x* or higher

The HTML documentation is available on the Web site:

https://www.xmlconverters.com/documentations

You can access the help system by navigating to the help subdirectory of the product installation directory and opening the following file from within your browser:

install_dir/help/help.htm

After the browser opens, the left pane displays the Table of Contents, Index, and Search tabs for the entire documentation library. When you have opened the main screen of the help system in your browser, you can bookmark it in the browser for quick access later.

NOTE: Security features set in your browser can prevent the Help system from launching. A security warning message is displayed. Often, the warning message provides instructions for unblocking the Help system for the current session. To allow the Help system to launch without encountering a security warning message, the security settings in your browser can be modified. Check with your system administrator before disabling any security features.

PDF Version

The product documentation is also provided in PDF format. You can view or print the documentation, and perform text searches in the files. The PDF documentation is available on the Web site:

https://www.xmlconverters.com/documentations

Javadoc

Stylus XML Converters provides Javadoc for the XML Converters packages. This system is installed when you install Stylus XML Converters.

Files for the Stylus XML Converters Javadoc are written to the javadoc directory where you installed Stylus XML Converters.

The Javadoc documentation is available at the following URL:

https://www.xmlconverters.com/documentations/javadoc

Contacting Technical Support

IVI Technologies offers a variety of options to meet your technical support needs. Please visit our Web site for more details and for contact information:

https://www.xmlconverters.com/support

The IVI Technologies Web site provides the latest support information through our global service network. The SupportNow program provides access to support contact details, tools, patches, and valuable information, including a list of FAQs for each product. In addition, you can search our Knowledge-base for technical bulletins and other information.

When you contact us for assistance, please provide the following information:

- Your customer number or the serial number that corresponds to the product for which you are seeking support, or a case number if you have been provided one for your issue.
- Your name, phone number, email address, and organization. For a first-time call, you may be asked for full customer information, including location.
- The IVI Technologies product and the version that you are using.
- The type and version of the operating system where you have installed your product.
- Any database, database version, third-party software, or other environment information required to understand the problem.
- A brief description of the problem, including, but not limited to, any error messages you have received, what steps you followed prior to the initial occurrence of the problem, any trace logs capturing the issue, and so on. Depending on the complexity of the problem, you may be asked to submit an example or reproducible application so that the issue can be re-created.

- A description of what you have attempted to resolve the issue. If you have researched your issue on Web search engines, our Knowledge-base, or have tested additional configurations, applications, or other vendor products, you will want to carefully note everything you have already attempted.
- A simple assessment of how the severity of the issue is impacting your organization.

June 2024, Release 6.2 of Stylus XML Converters

1 Overview

Stylus XML Converters is a library of Java classes that provides programmatic bi-directional access to numerous data sources such as EDI, CSV, and other legacy formats such as XML through Java applications.

This chapter provides an overview of the Stylus XML Converters, including the types of file formats that are supported, how they can be used to access data from other sources, and examples of converting EDI to XML and XML Schema generation from EDI.

Types of Stylus XML Converters®

Stylus XML Converters support numerous file formats, from many EDI dialects to common formats such as CSV and tab-delimited files. Most Stylus XML Converters are bidirectional, allowing you to convert from a native format to XML and vice versa.

Table 1-1 summarizes the types of file formats supported by the Stylus XML Converters and indicates which converters are bidirectional.

Table 1-1. File Formats Supported by Stylus XML Converters

File Type	Description	Bidirectional
Base-64	Converts any file, text or binary (such as an image), into an XML document with a single element containing the Base-64 encoded content of the input file.	Yes
Binary	Similar to the Base-64 XML Converter, except with hexadecimal output. Other options allow output in other bases, such as decimal, octal, or binary.	Yes
CSV	Converter for comma-separated values (CSV) files. Supports multiple encodings and options to tune the quote and escape characters. Supports multiple delimiters including commas.	Yes
dBase	Support for dBase II, III, III+, IV, and V formats.	Yes
DIF	Data Interchange Format (DIF) is a spreadsheet-based file format. There are also XML Converters for Super Data Interchange (SDI) and Symbolic Link (SYLK).	Yes
DotD	Support for the Progress OpenEdge® text dump file format.	Yes
EDI	Automatically detects and parses ACORD AL3, EANCOM, EDIFACT, Edig@s, HIPAA, HL7, IATA AHM780, IATA Cargo-IMP, IATA PADIS, NCPDP SCRIPT, NCPDP Telecommunication, TRADACOMS, and X12 EDI message types, with options for custom message types and message extensions to cover proprietary EDI-based formats.	Yes
HTML	Support for HTML to XML (XHTML) file conversion.	Yes
JavaProps	Support for the Java .properties file format, which is used for program configuration, translation, and data storage.	Yes
JSON	Uses the algorithms on the JSON.org website to read from XML and write to JSON (JavaScript Object Notation), and vice-versa.	Yes
Line	Reads in text one line at a time, wrapping an element around each line and escaping any embedded &, >, or < symbols.	Yes

Table 1-1. File Formats Supported by Stylus XML Converters

File Type	Description	Bidirectional
Pyx	Support for the PYX line-oriented notation for expressing tree-oriented data.	Yes
RTF	Converts rich-text format (RTF) into XML, and vice versa.	Yes
SDI	Super Data Interchange (SDI) is another popular spreadsheet-based file format. There are also Stylus XML Converters for DIF and SYLK.	Yes
SYLK	Symbolic Link (SYLK) is another popular spreadsheet-based file format. There are also Stylus XML Converters for DIF and SDI.	Yes
TAB	Tab-separated values format commonly associated with Microsoft Excel spreadsheets.	Yes
WinIni	Converter for Windows .ini configuration files.	Yes
WinWrite	Converter for Microsoft WinWrite files; renders XHTML.	No
Custom	Custom XML converters (.conv files) created using Stylus Studio.	No

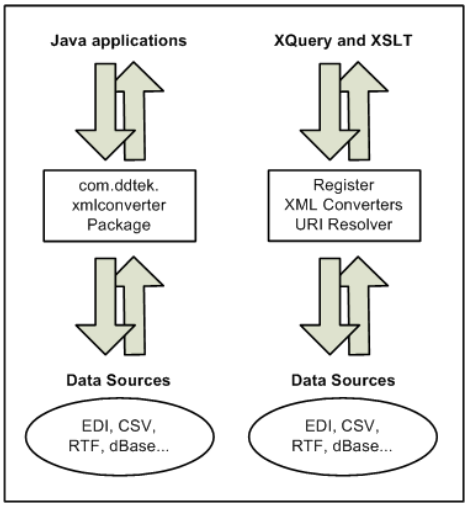
Customizing Stylus XML Converters®

Each Stylus XML Converters has properties that allow you to customize the converter to suit your needs. Some converters, for example, let you specify the line separator character, escape character, root element name, and other aspects of the output format. Default values are used for all properties that you do not explicitly specify. You specify properties in the converter: URI string that you use to invoke the converter.

See “URI Schemes” on page 22 to learn more about the converter: URI. See Chapter 5 “Stylus XML Converters® Properties” on page 113 for a description of all supported properties.

Data Access

Stylus XML Converters provide access to non-XML data stored in EDI and flat file formats such as CSV, RTF, dBase, binary, and others. The following figure illustrates the different ways that you can use Stylus XML Converters to access XML and non-XML data from Java applications and from XQuery or XSLT code.



Data access to non-XML data stored as EDI or in another file format (CSV or tab-delimited, for example) is accomplished using the converter: URI scheme. See “URI Schemes” on page 22 to learn more about the URI schemes supported by Stylus XML Converters.

URI Schemes

In Java, files and other data resources are referenced using the file:, http:, ftp:, and a limited set of other URI schemes.

The Stylus XML Converters Java API extends the functionality of the basic URI to recognize and understand the converter: URI scheme developed by IVI Technologies. You can use the converter: URI scheme in your Java, XQuery, and XSLT code.

For example, the following URI scheme invokes the custom XML conversion of the `myConverter.conv` file:

```
converter:myConverter.conv
```

The following URI scheme invokes the EDI XML Converter, using the editeur.edi file as the EDI source to be converted:

```
converter:EDI?file:///m:/testing/editeur.edi
```

See Chapter 2 “Stylus XML Converters® URI Schemes” on page 53 for more information.

Command-Line Usage

You can run Stylus XML Converters from the command line.

To specify a native file to be converted to XML:

```
java -jar xmlconverters.jar /to [/analyze] [/report filename]
/converter name[:property_name=value ...] /in filename [/out filename]
```

To specify an XML file to be converted to a native format:

```
java -jar xmlconverters.jar /from /converter name[:property_name=value ...]
/in filename [/out filename]
```

Usage Notes

The following list provides some usage notes for running Stylus XML Converters from the command line:

- The `/to` option specifies that you are converting a file from its native format to XML; the `/from` option specifies that you are converting an XML file to the file type specified in the `/converter` option.
- The XML Converter specified in the *`name`* argument for the `converter` option accepts settings for properties specific to that converter. For example, `/converter EDI:newline=cr` indicates that the carriage return (`cr`) is to be used as the line separator (`newline`) character.
- *`property_name=value`* pairs cannot include blanks. For example, `newline=platform` is valid and `newline = platform` is not valid.
- Use `/analyze` to analyze and convert an EDI stream; use `/report` if you want to save the analysis report (by default, the report is written to a temp file and deleted after the conversion). The `/analyze` and `/report` options can only be used when converting EDI to XML (that is, when you are using the `/to` option.

See Chapter 3, “Analyzing EDI-to-XML Conversions” for more information.

- Use `/in -` to read from the standard input.
- To write to the standard output, omit the `/out` option.
- You can use dashes (-) instead of forward slashes (/) to separate options (for example, `-to` instead of `/to`).
- To generate an XML Schema, replace the `/to` option with the `/schema` option. See “XML Schema Generation” on page 30 for more information.
- To generate HTML or XHTML documentation, replace the `/to` option with the `/html` option. See “HTML/XHTML Documentation Generation” on page 41 for more information.
- To generate sample output, replace the `/to` option with the `/sample` option and specify either the `/xml` or `/edi` option to determine the format for the output. See “Sample Output Generation” on page 47 for more information.

Example

The following example uses the EDI XML Converter to convert the input file (831.x12) to an XML file (my831.xml) using the default values for the EDI XML Converter:

```
java -jar xmlconverters.jar /to /converter EDI /in ..\examples\831.x12
/out my831.xml
```

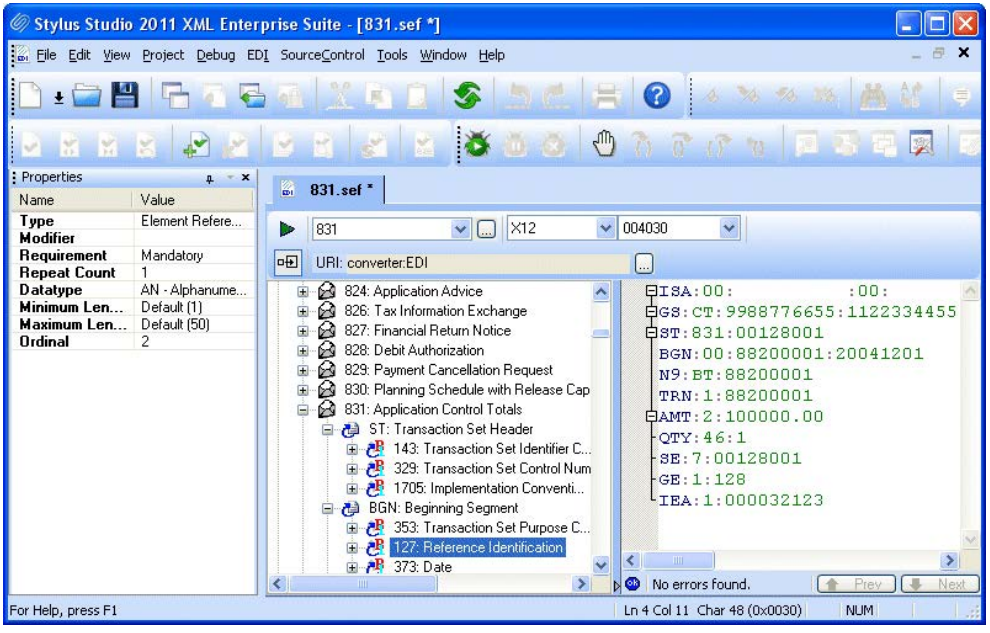
The 831.x12 sample file and others are in the *installdir*\examples directory, where *installdir* is your product installation directory. To learn more about the examples, see “Example Applications” on page 50.

Handling Proprietary EDI Formats

Stylus XML Converters supports the Standard Exchange Format (SEF). SEF allows you to specify an EDI structure, typically one that differs from one of the EDI standards such as EDIFACT or X12. You save this structure definition in an SEF file, which you can then instruct the EDI XML Converter to use when converting proprietary EDI to XML.

Creating an SEF File

You can create SEF files manually based on the SEF specification. This process, however, can be difficult and error-prone. An easier way is to use the Stylus Studio XML Enterprise EDI to XML Module, which provides a visual editor to help you build SEF files based on standard EDI dialects.



When using the Stylus Studio EDI to XML Module, you can choose as your starting point an EDI document (from which an EDI dialect is inferred) or an EDI dialect. From there, you use the EDI to XML Module editor's tools to define the ways in which your proprietary EDI structure differs from the EDI standard.

The SEF Specification

You can find the SEF specification on the following Web site:

https://www.xmlconverters.com/standards/sef

Example: Using an SEF File

See "Example 7" on page 98 for an example of using an SEF extension file to define an XML Schema. See also "Loading SEF Files Programmatically" on page 110 for information about loading SEF files

Managing Errors

The Stylus XML Converters API provides the following ways to manage errors in your applications:

- EDI Analyzer
- ConverterListener Interface
- EDIConverterListener Interface
- EDIConverterException Interface

NOTE: These features are implemented only for the EDI XML Converter.

EDI Analyzer

The EDI Analyzer API allows you to analyze an EDI stream for errors that can cause the XML Converter to throw an exception before converting the EDI stream to XML. A report generated by the EDI Analyzer in XML format identifies and describes any errors. The EDI Analyzer also automatically generates Accept/Reject messages that can be forwarded to the EDI sender.

The EDI Analyzer API is supported for EDI-to-XML conversions only (not vice versa).

See Chapter 3, “Analyzing EDI-to-XML Conversions” for more information on the EDI Analyzer API.

ConverterListener Interface

In an application, it is not always necessary for warnings and errors to throw exceptions and stop the conversion process. You may want to simply make the application aware that a problem has occurred and allow it to recover (or not) from the warning or the error.

The Converter Listener interface allows you to intercept warnings, errors, and fatal errors and manage them separately. The default action is to ignore warnings and to throw exceptions for errors and fatal errors.

Processing can resume after both warnings and errors. For example, if an exception is not thrown, processing continues. If an error occurs, it is possible that other errors will cascade from the first. Fatal errors can be reported, but on their return, a ConverterException is always thrown by the EDI XML Converter engine. The exception that is thrown is an instance of ConverterException or one of its subclasses such as ConverterArgumentException.

Example

See “Example 8” on page 99 for an example of registering a ConverterListener.

EDIConverterListener Interface

The EDIConverterListener is a specialized version of ConverterListener; its methods provide detailed information about error conditions. The invalidCharacter() method, for example, is called when a character does not match the specified encoding in the EDI stream. Similarly, unknownCodeListValue() is called when a codelist validation fails.

EDIConverterException Interface

Typically, EDI-based conversions are more complex than other types of conversions (those for CSV and tab-delimited files, for example). By providing more contextual information about where a problem has occurred, the EDI XML Converter allows you to capture the error and return standard EDI messages back to the sender of the message. For example, you can return CONTRL (for EDIFACT), 997 (for X12), 999 (for HIPAA), or ACK (for HL7).

The EDIConverterException is a specialized version of ConverterException that contains extra information about the context of errors in EDI files. When a ConverterException is thrown while processing an EDI file, or when a ConverterListener is registered and a warning(), error(), or fatalError() is called, the exception that is thrown is probably EDIConverterException. This exception contains the following methods to probe the context of the specific error – GetContentData(), getControlData(), getData(), and getError(). Processing can recover from both warnings and errors; however, fatal errors always stop the processing.

Error Diagnostics

Full context information is provided when an error is encountered in the EDI file, including the error number according to the local EDI dialect. For example, many EDIFACT errors are recorded in the 0085 element codelist, and if an error matches one of those, it is reported as such.

XML Schema Generation

You can use the SchemaGenerator interface to create XML Schemas that describe the structure of XML files that are read or created by a ConvertToXML or ConvertFromXML object. You may want to use the SchemaGenerator interface in the following situations:

- You have a fromXML converter: URI, and you want to know the XML Schema that the input XML data must satisfy.
- You have a toXML converter: URI, and you want to know the XML Schema of the XML output.
- You have a toXML converter: URI and a non-XML data file, and you want to know the XML Schema of the XML output.

  NOTE: This functionality is available only with certain XML Converters. See “XML Schema Generation Summary” on page 40 for more information.

The generated XML Schema depends on the type of file it is generated from, and not on the actual data. For example, if you are generating an XML Schema for an EDI file, the EDI XML Converter is concerned only with the file’s dialect, version, and message type/transaction set. You can specify this information by providing a sample file input or by specifying the appropriate properties in the converter: URI. See “Instance Documents” on page 33 and “Converter URI Properties” on page 33 for more information.

Command-Line Usage

To generate an XML Schema from the command line:

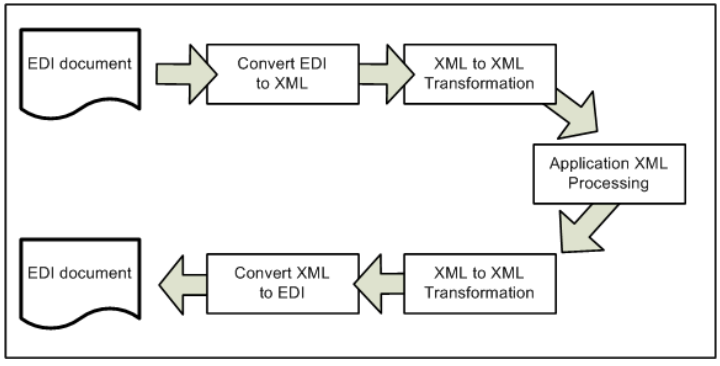
```
java -jar xmlconverters.jar /schema /converter name
[:property_name=value [:property_name=value ...] ] /in filename
[/out filename]
```

Some converters (such as the EDI XML Converter) require that you provide an instance document or that you specify sufficient properties in the converter: URI. For the EDI XML Converter, for example, you must provide the dialect, version, and message type/transaction set.

Example Scenario

Your company routinely receives client data in EDI files. This data must be converted to XML so that it can be transformed for processing by an application. After application processing, the resulting XML is again transformed to another format before being converted back to EDI.

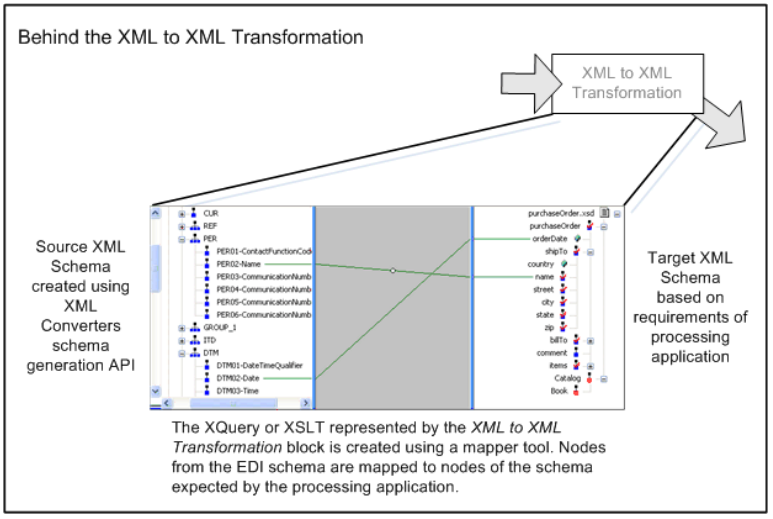
The following illustration shows the workflow:



In this illustration, the conversion block *Convert EDI to XML* represents an instance of the EDI XML Converter, which is used to convert the incoming EDI document, which may be an X12 810

transaction set (Invoice). This XML conforms to the XML Schema consistent with the X12 EDI transaction set from which it was derived. Before the XML can be used by the application, it must be transformed into the format expected by the application – that is, it must conform to an XML Schema.

Suppose that XQuery, using the Stylus XQuery® implementation, for example, is used to perform this transformation (see the preceding illustration). One way to create an XQuery is to use a mapping tool. You can map nodes from the XML Schema representing the EDI X12 810 transaction set to the XML Schema representing the document format expected by the XML processing application. You can use the EDI XML Converter to create the XML Schema for the EDI X12 810 transaction set, as shown in the following illustration.



A similar mapping process is performed to create the second XML-to-XML transformation (another XQuery), mapping XML Schema nodes from the application format to the EDI X12 810 transaction set format to create an XQuery. This XQuery transforms the XML data to a format that can be understood by the EDI XML Converter.

See “Creating XML Schemas from EDI” on page 51 for an example of using the EDI XML Converter to create an XML Schema from an EDI

document. To learn more aboutStylus XQuery, see “More About Stylus XQuery®” on page 59.

Instance Documents

Some Stylus XML Converters, such as those for CSV and Tab, require an instance document to provide the converter with the information it needs to generate an XML Schema. Other converters (such as the EDI XML Converter) can use instance documents, but they are not required. You can also provide information using converter: URI properties to specify characteristics of the generated XML Schema. Still others (Base64 and SDI, for example) use neither instance documents nor converter: URI properties, relying instead on the built-in settings of the converter for XML Schema generation.

See “XML Schema Generation Summary” on page 40 for more information concerning instance document and converter: URI property usage.

Converter URI Properties

This section describes how URI properties affect XML Schema generation for the XML Converters that support it. The Stylus XML Converters that you can specify URI properties for are:

- CSV
- EDI
- Line
- Tab
- *custom* (built using Stylus Studio)

CSV XML Converter URI Properties

The properties listed in Table 1-2 affect XML Schema generation for both CSV and tab-delimited files.

Table 1-2. Properties for the CSV and Tab XML Converters

Property	Description
first=	Specifies whether elements subordinate to the row element are named column (plus a number to make it unique) or are given their name based on the first row of data.
root=	Specifies the name of the root element in the converted XML; also specifies the name of the root element in the generated XML Schema.
row=	Specifies the name of the row element in the converted XML; also specifies the name of the row element in the generated XML Schema.

EDI XML Converter URI Properties

The properties listed in Table 1-3 affect XML Schema generation for EDI files.

Table 1-3. Properties for the EDI XML Converter

Property	Description
dialect=	Specifies the EDI dialect. Valid values: AHM780, CARGO, EANCOM, EDIFACT, EDIGAS, HIPAA, HL7, IATA, NCPDP, TELCO, TRADACOMS, or X12 Use IATA to specify the PADIS dialect. The dialect must be specified if an instance document is not provided.
doc=	Determines whether to include xs:documentation comments in the XML Schema. n yes – comments are included in the XML schema. n no – comments are not included in the XML schema. The default is yes.

Table 1-3. Properties for the EDI XML Converter

Property	Description
hipaa=	Determines whether the converter should determine if an X12 file is a HIPAA file. If the file is a HIPAA file, HIPAA rules are used; otherwise, X12 rules are used. Valid values: n yes – the converter determines whether the X12 file is a HIPAA file. n no – the converter processes the file as an X12 document, even if it is recognized as a HIPAA document. n loop – same as yes, but this value also creates a nested loop structure for the converted XML and generated XML Schema, which can simplify this output's use in XML mapping tools. The default is no.
inter=	Certain EDI messages have alternate batch and interactive forms, depending upon whether they are used between systems that have real-time connections. Valid values: n yes – the interactive form is used, if available. For example, in EDIFACT, the normal envelope of UNB/UNH/UNT/UNZ would be replaced by UIB/UIH/UIT/UIZ. n no – the interactive form is not used. The default is yes.

Table 1-3. Properties for the EDI XML Converter

Property	Description
long=	Determines whether to use long or short element and/or segment names in your XML schema generation (FTX03-TextReference or FTX03, for example). Valid values: n elements – long names are used for elements. (In previous versions, long=yes could be used. For naming consistency, long=yes has been deprecated.) n segments – long names are used for segments. n all – long names are used for both elements and segments. n none – abbreviated names are used for both. The default is none.
message=	Varies based on the dialect and version. Examples for EDIFACT include CONTRL and ORDERS; examples for HL7 include ACK and ADT_A01; examples for IATA PADIS include SPORES and TKTRES; and so on. If an instance document is not provided, this property is required.
syntax=	Sets the syntax level for the EDIFACT family of EDI dialects (including Edig@s, IATA, EANCOM, and NCPDP). Valid values are 1, 2, 3 and 4. The default varies based on the dialect and version.

Table 1-3. Properties for the EDI XML Converter

Property	Description
tbl=	Determines whether the codelist tables are created as enumerations in the generated XSD output. Valid values: n yes – codelist tables are created as enumerations in the generated XSD output. n no – codelist tables are not created as enumerations in the generated XSD output. The default is no.
user=	Specifies an SEF extension file. The structure of the specified file is incorporated in the generated XML Schema.
version=	Varies based on the dialect. Examples for EDIFACT include 921 and D07A; examples for HL7 include 2.1 and 2.5; examples for IATA PADIS include 99-1 and 99-2; and so on. If an instance document is not provided, this property is required.

Line XML Converter URI Properties

The properties listed in Table 1-4 affect XML Schema generation for whole-line text files:

Table 1-4. Properties for the Line XML Converter

Property	Description
line=	Specifies the name of the element that wraps each line in the converted XML. Also specifies the name of that element in the generated XML Schema.
root=	Specifies the name of the root element in the converted XML. Also specifies the name of the root element in the generated XML Schema.

Tab XML Converter URI Properties

See “CSV XML Converter URI Properties” on page 34.

XML Schema Generation Summary

Table 1-5 summarizes information about the XML Schema generation capabilities of the converters that support generating an XML Schema, indicates whether an Instance document is required, and indicates whether you can specify URI properties that affect the XML schema generation.

Table 1-5. Stylus XML Converters That Support Generating an XML Schema

Converter Name	Instance Document Required?	URI Properties that Affect XML Schema?
Base64	No	No
Binary	No	No
CSV	Yes	Yes
custom	No	via.conv file
dBase (all)	Yes	No
DIF	No	No
DotD	No	No
EDI (all)	Optional[1]	Yes
JavaProps	No	No
Line	No	Yes
SDI	No	No
Sylk	No	No
Tab	Yes	Yes
WinIni	No	No
WinWrite	No	No

1. Optional means that an instance document is used if provided, but one is not required.

HTML/XHTML Documentation Generation

Using the EDI XML Converter, you can generate HTML/XHTML documentation for all supported EDI dialects and transactions.

Command Line Usage

To generate HTML/XHTML documentation from the command line, use the following syntax:

```
java -jar xmlconverters.jar /html /converter name [:property_name=value
[:property_name=value ...] ] /in filename [/out filename]
```

The EDI XML Converter requires that you provide an instance document or specify the following properties in the converter URI: dialect, user, version, and message type/transaction set.

Converter URI Properties

Table 1-6 describes the properties of the EDI XML Converter that affect HTML/XHTML document generation. See “EDI XML Converter Properties” on page 126 for a complete list of properties supported by the EDI XML Converter.

Table 1-6. Properties for the EDI XML Converter: HTML/XHTML Document Generation

Property	Description
dialect=	Specifies the EDI dialect. Valid values: AHM780, CARGO, EANCOM, EDIFACT, EDIGAS, HIPAA, HL7, IATA, NCPDP, TELCO, TRADACOMS, or X12 Use IATA to specify the PADIS dialect. If an instance document is not provided, this property is required.
doc=	Determines whether to include xs:documentation comments in the XML Schema. The default is yes.
hipaa=	Determines whether the converter should determine if an X12 file is a HIPAA file. If the file is a HIPAA file, HIPAA rules are used; otherwise, X12 rules are used. Valid values: n yes – The converter determines whether the X12 file is a HIPAA file. n no – The converter processes the file as an X12 document, even if it is recognized as a HIPAA document. n loop – same as yes, but this value also creates a nested loop structure for the converted XML and generated XML Schema, which can simplify this output's use in XML mapping tools. The default is no.

Table 1-6. Properties for the EDI XML Converter: HTML/XHTML Document Generation

Property	Description
inter=	Certain EDI messages have alternate batch and interactive forms, depending on whether they are used between systems that have real-time connections. Valid values: n yes – the interactive form is used, if available. For example, in EDIFACT, this would cause the normal envelope of UNB/UNH/UNT/UNZ to be replaced by UIB/UIH/UIT/UIZ. n no – the alternate batch form is used. The default is no.
long=	Determines whether to use long or short element and/or segment names in your XHTML document generation (FTX03-TextReference or FTX03, for example). Valid values: n elements – long names are used for elements. n segments – long names are used for segments. n all – long names are used for both elements and segments. n none – abbreviated names are used for both. The default is none.
message=	Varies based on the dialect and version. Examples for EDIFACT include CONTRL and ORDERS; examples for HL7 include ACK and ADT_A01; examples for IATA PADIS include SPORES and TKTRES; and so on. If an instance document is not provided, this property is required.

Table 1-6. Properties for the EDI XML Converter: HTML/XHTML Document Generation

Property	Description
syntax=	Sets the syntax level for the EDIFACT family of EDI dialects (including Edig@s, IATA, EANCOM, and NCPDP). Valid values are 1, 2, 3 and 4. The default varies based on the dialect and version.
tbl=	Determines whether the codelist tables are created as enumerations in the generated XSD output. The default is no.
user=	Specifies an SEF extension file. If an instance document is not provided, this property is required.
version=	Varies based on the dialect. Examples for EDIFACT include 921 and D07A; examples for HL7 include 2.1 and 2.5; examples for IATA PADIS include 99-1 and 99-2; and so on. If an instance document is not provided, this property is required.

Example Scenario

Suppose an insurance company implements a Web application that allows its business partners to file claims through EDI transactions. In addition to generating an error message when a claim is submitted with missing information or information in the incorrect format, the Web application uses the XML EDI Converter to generate an HTML document that describes the correct format of the information that is required.

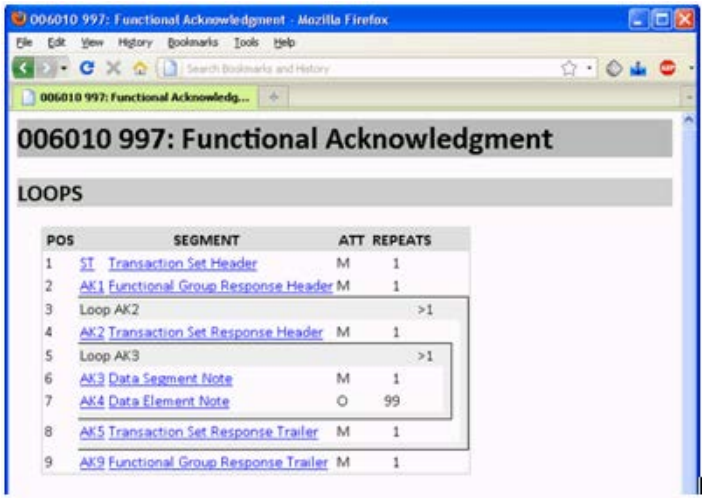
For example, the Web application can generate an HTML document by executing the following command:

```
java -cp XMLConverters.jar com.ddtek.xmlconverter.CmdLine /html /converter
EDI:dialect=X12:message=997:version=006010:tbl=yes /out 006010-997.html
```

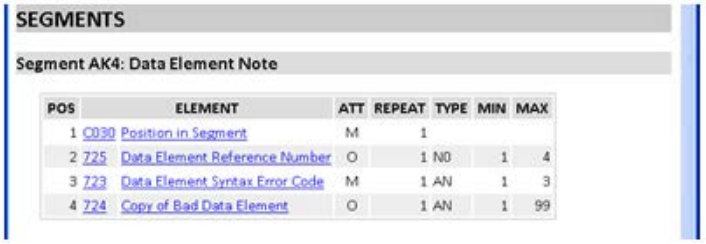
As shown in the following examples, the documentation that is produced is divided into four sections: message, segments, composites, and elements.

The first section describes the message looping structure and contains hyperlinks to segments, composites, and elements.

Special types of loops, such as Interleaved or Unique loops, are indicated. By default, loop names are shown.



The second section shows the segments and contains hyperlinks to composites and elements. If an element uses an internal code list subset, that list is attached to the element.

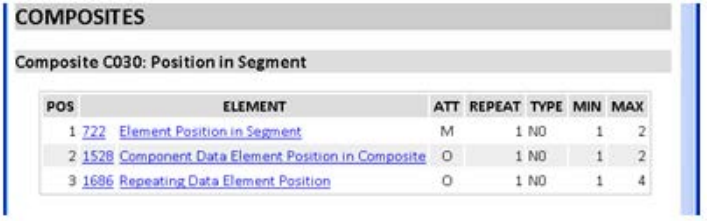


SEGMENTS

Segment AK4: Data Element Note

POS	ELEMENT	ATT	REPEAT	TYPE	MIN	MAX
1	C030 Position in Segment	M	1			
2	725 Data Element Reference Number	O	1	N0	1	4
3	723 Data Element Syntax Error Code	M	1	AN	1	3
4	724 Copy of Bad Data Element	O	1	AN	1	99

The third section shows the composites and contains hyperlinks to elements. If a dialect, such as HL7, has nested composites, they are also listed.

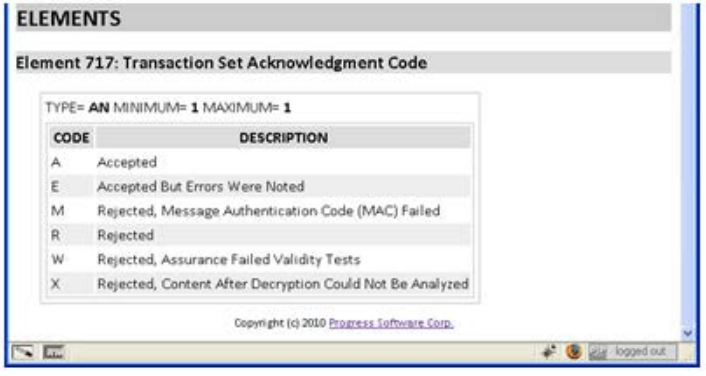


COMPOSITES

Composite C030: Position in Segment

POS	ELEMENT	ATT	REPEAT	TYPE	MIN	MAX
1	722 Element Position in Segment	M	1	N0	1	2
2	1528 Component Data Element Position in Composite	O	1	N0	1	2
3	1686 Repeating Data Element Position	O	1	N0	1	4

The fourth section shows the elements and any associated code lists.



ELEMENTS

Element 717: Transaction Set Acknowledgment Code

TYPE= **AN** MINIMUM= **1** MAXIMUM= **1**

CODE	DESCRIPTION
A	Accepted
E	Accepted But Errors Were Noted
M	Rejected, Message Authentication Code (MAC) Failed
R	Rejected
W	Rejected, Assurance Failed Validity Tests
X	Rejected, Content After Decryption Could Not Be Analyzed

Sample Output Generation

Using the EDI XML Converter, you can generate sample output for a specific transaction in either an XML or EDI format. You can create a range of samples—from the minimum set of required elements to a complete EDI transaction, including optional elements. The EDI XML Converter can also generate random values according to the field type and range, which allows you to test transactions and simulate user data without opening up actual data to a possible security risk.

Command Line Usage

To generate sample output from the command line, use the following syntax:

```
java -jar xmlconverters.jar /sample converter:EDI [:property_name=value
[:property_name=value ...] ] /in filename [/out filename]
```

The EDI XML Converter requires that you provide an instance document or specify the following properties in the converter URI: dialect, version, and message.

Converter URI Properties

Table 1-7 describes the properties of the EDI XML Converter that affect sample output generation. See “EDI XML Converter Properties” on page 126 for a complete list of properties supported by the EDI XML Converter.

Table 1-7. Properties for the EDI XML Converter: Sample Output Generation

Property	Description
data=	Controls whether sample data is generated with XML and EDI sample output files. Valid values: n none – No sample data is generated. Headers and trailers are generated. n minimal – The minimum amount of data required to validate the sample file is generated. n random – Random strings are generated; codelist values are selected randomly to fill the data. The default is none.
emit=	Controls which optional segments are generated in XML and EDI sample output files. Valid values are: n none – Only mandatory segments and elements are created. n segments – All segments are created, but only mandatory elements within those segments are populated with sample data. n elements – All segments and all elements are created. Some elements may be populated with sample data and some may be empty. The default is none.

Example Scenario

Suppose your company needs to transition from an earlier HIPAA standard to the latest HIPAA standard. To fully test your system before it goes into production, you need realistic test data. Creating user data can be time-consuming. In addition, duplicating the actual user data opens up that data to a possible security risk. Instead, you can use the EDI XML Converter to easily generate sample output and supply random values for data in the output.

For example, suppose you execute the following command:

```
java -cp XMLConverters.jar com.ddtek.xmlconverter.CmdLine /sample /edi
/converter EDI:message=997:version=004010:dialect=X12:data=random
```

The sample output in EDI format that is produced would look similar to the following example:

```
ISA+00+ +00+ +ZZ+ISENDER +ZZ+IRECEIVER8
   +100610+0336+U+00401+999999999+0+T+:'
GS+FA+ASENDER+ARECEIVER+20100210+1524+888888888+X+004010'
ST+997+777777777'
AK1+VA+1'
AK9+M+1+1+1'
SE+4+777777777'
GE+1+888888888'
IEA+1+999999999'
```

Example Applications

This section shows two simple applications: one showing the conversion of an EDI file to XML and another showing how to use the API to create an XML Schema from an EDI file.

See Chapter 4 “Stylus XML Converters® Examples” on page 87 for more application examples.

Converting EDI to XML

Here is a simple example application that reads EDI from one file (myEdi.x12) and writes XML to another (myEdi.x12.xml).

```
import com.ddtek.xmlconverter.*;
import javax.xml.transform.stream.*;

public class ConverterOne {
   public static void main(String args[]) throws Throwable {

   System.out.println(args[0] + " --> " + args[1]);
   Converter toXML =
ConverterFactory.newInstance().newConvertToXML("converter:EDI");
    toXML.convert(new StreamSource(args[0]), new StreamResult(args[1]));
   }
}
```

This program can be invoked from a command line as shown:

```
java ConverterOne file:///c:/path/myEdi.x12  file:///c:/path/myEdi.x12.xml
```

Creating XML Schemas from EDI

Here is a simple example that generates an XML Schema for EDIFACT version D07A. The name of the message being converted (in this case, ORDERS), is taken from the command line, which might look like this:

```
java com.ddtek.example.CreateEdifactSchema ORDERS
```

In this example, the XML Schema is written to the console.

```
package com.ddtek.example;
import javax.xml.transform.stream.StreamResult;
import com.ddtek.xmlconverter.ConverterFactory;
import com.ddtek.xmlconverter.SchemaGenerator;
import com.ddtek.xmlconverter.exception.ConverterException;
public class CreateEdifactSchema {
   public static void main(String[] args) {
      String uri = "EDI:dialect=EDIFACT:version=D07A:long=elements:message="
         + args[0];
         try {
            ConverterFactory factory = new ConverterFactory();
            SchemaGenerator schema = factory.newSchemaGenerator(uri);
            StreamResult sr = new StreamResult(System.out);
            schema.getSchema(sr);
         catch (ConverterException ce) {
         ce.printStackTrace();
         }
   }
```

The previous example specified the dialect, version and message directly in the EDI: URI using the dialect=, version= and message= properties:

```
...
String uri = "EDI:dialect=EDIFACT:version=D07A:long=elements:message=" +
args[0];
...
```

For some file types, such as EDI, you can supply a sample, or instance, document from which the converter engine can read this information. When you use an EDI instance document, the schema generator generates an XML Schema for the dialect/version/message in that instance document.

In the following example, the name of the EDI instance document (data.edi) is derived from the command line, which might look like this:

```
java com.ddtek.example.CreateAnySchema c:\myhome\data.edi
```

In this example, the XML Schema is written to the console:

```
package com.ddtek.example;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;
import com.ddtek.xmlconverter.ConverterFactory;
import com.ddtek.xmlconverter.SchemaGenerator;
import com.ddtek.xmlconverter.exception.ConverterException;
public class CreateAnySchema {
   public static void main(String[] args) {
      String uri = "EDI:long=elements";
      try {
         ConverterFactory factory = new ConverterFactory();
         SchemaGenerator schema = factory.newSchemaGenerator(uri);
         StreamSource ss = new StreamSource(args[0]);
         StreamResult sr = new StreamResult(System.out);
         schema.getSchema(ss, sr);
      } catch (ConverterException ce) {
      ce.printStackTrace();
      }
   }
}
```

2 Stylus XML Converters® URI Schemes

You can use the converter: URI scheme to reach a variety of data sources using Stylus XML Converters. The converter: URI scheme also can be used with user-defined custom XML conversions created using Stylus Studio XML Enterprise Suite.

The converter: URI Scheme

The converter: URI scheme specifies one of the following converter names:

- One of the standard Stylus XML Converters (for EDI or tab-delimited files, for example) and settings for the converter's properties
- A custom XML conversion created using Stylus Studio XML Enterprise Suite

SEF files, which describe proprietary extensions to EDI standard dialects and messages, can be passed as a property of the converter: URI.

Converter URI Syntax

Although properties differ from one XML Converter to the next, the syntax used to invoke an XML Converter is the same:

`converter:`*`name`*`[:`*`property_name`*`=`*`value`*`]... [?`*`URI`*`]`

When you specify a converter: URI, you identify:

- The converter you want to use (EDI, CSV, dBase, and so on)
- Options for the specified converter (separator and escape characters, for example)
- The file to be converted
- The direction of the conversion (for example, from XML or to XML)

Syntax Validation

The converter: URI syntax is validated at runtime. Exceptions are thrown in the following situations:

- If an unknown property is specified (for example, `converter:EDI:myproperty=yes`)
- If the converter: URI includes a dialect and a property that is not valid for that dialect.
- When a property that has values that represent a choice (for example, yes|no|always) is used with an invalid value (for example, `auto=xyz`). Note that valid values for booleans start with y, n, t, f, 0, or 1.

If the converter: URI detects a valid property that does not apply to the operation being performed, that property is ignored.

Example

The following converter: URI invokes the converter for comma-separated values to convert the three.txt file located in the <install dir>\examples\ directory to XML:

```
converter:CSV:newline=lf:first=yes?file:///c:/XMLConverters/examples/three.txt
```

The following instructions are sent to the converter engine from this instance of the converter URI:

- Use the Comma-Separated Values XML Converter (`converter:CSV`).
- The line separator in the source file is a line feed (`newline=lf`).
- The values in the first row of the source file are used to supply field names (`first=yes`).
- The source file is three.txt (`?file:///c:/XMLConverters/examples/three.txt`).

In this example:

- The name of the XML Converter is CSV. It could be any converter – EDI, Base64, DIF, RTF, and so on.
- Only the newline= and first= properties are specified. Default values are used for all other converter properties.
- The source file being converted is three.txt, which is located in the /XMLConverters/examples directory. If you are using the converter: URI programmatically, omit the ? URI property because the source file is specified by the application.

Specifying XML Converter Properties

XML Converter properties that use default values do not have to be specified in the converter URI. A comma is the default separator

character for the CSV XML Converter, for example. If the particular file you are converting used another separator character, you would need to specify it using the sep= property.

Although the basic format of the converter URI is the same from one XML Converter to another, individual converters have different properties. For example, the XML Converter for dBase files has properties that the XML Converter for binary files does not.

See Chapter 5, “Stylus XML Converters® Properties” for a complete description of properties for all Stylus XML Converters.

Building a converter: URI

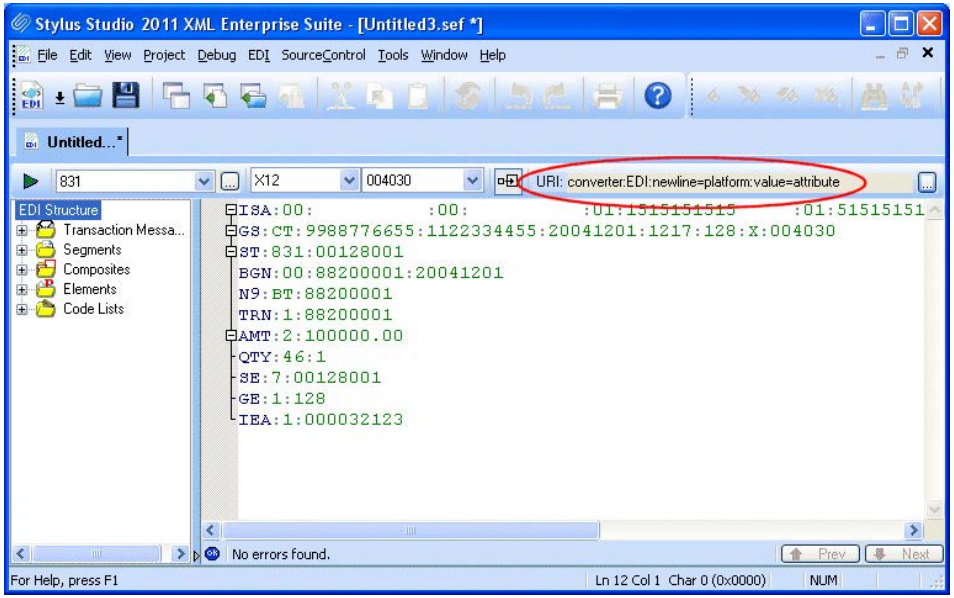
If you have Stylus Studio XML Enterprise Suite, you can use Stylus Studio to build the converter URIs for use in your Java applications. Converter URIs can be long and complex because properties and their values vary from one converter to another, so using Stylus Studio to construct them can reduce errors in your applicationsStylus Studio. Its GUI interface can make it easier to specify the converter properties you need to include.

Otherwise, you must construct the converter URL manually, taking care to specify both setting names and their values correctly. See Chapter 5, “Stylus XML Converters® Properties”for a complete description of properties for all Stylus XML Converters.

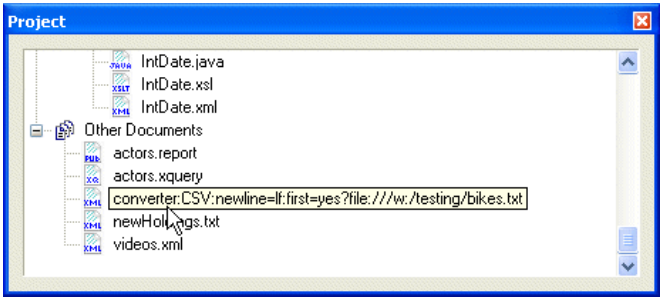
Converter URIs in Stylus Studio®

Converter URIs are displayed in the following places in Stylus Studio XML Enterprise Suite:

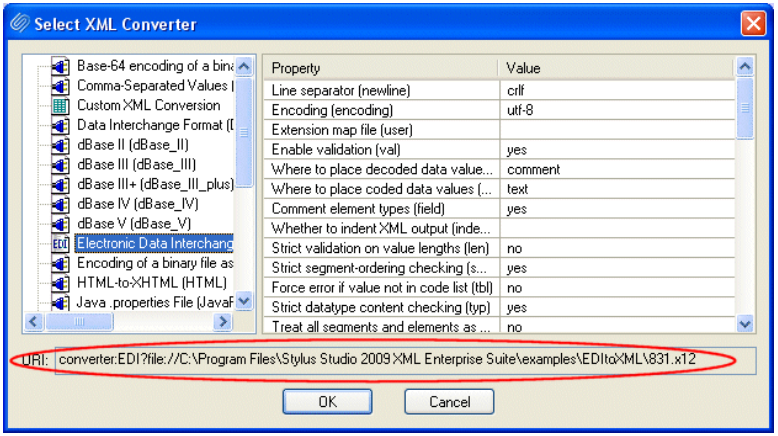
- In the URI field of the EDI to XML Module editor. You can also make changes to the converter URI here.



- In the Project window (select **Show Full URL Info** from the Project window shortcut menu)



- In the URI field of the Select XML Converter dialog box, as shown in the following illustration.



You can use any of these sources to capture the converter: URI string for use in your Java applications. For more information, refer to your Stylus Studio product documentation.

Invoking a Custom XML Conversion

The converter: URI scheme can be used to reference a custom XML conversion (a .conv file) built using Stylus Studio XML Enterprise Suite. The converter URI specifies only the location of the .conv file; the custom XML conversion contains all the information required to perform the XML conversion.

The following converter URI references a custom XML conversion named myConverter.conv:

```
converter:///myConverter.conv?file:inventory.txt
```

It uses myConverter.conv to convert the file inventory.txt to a format that is specified in the custom XML conversion when it was built using Stylus Studio XML Enterprise Suite.

NOTE: Custom XML conversions are defined using Stylus Studio XML Enterprise Suite.

Invoking a Converter URI in Stylus XQuery®

Stylus XQuery®, an XQuery implementation, uses a document URI resolver that enables the XQuery doc() function to take a converter: URI as its argument.

Consider the following example, which uses the doc() function to invoke the CSV XML Converter to convert the file request.csv to XML:

```
doc("converter:CSV:first=yes?request.csv")
```

In this example, only one of the CSV XML Converter properties is set (`first=yes`); default settings are used for all other properties.

More About Stylus XQuery®

Stylus XQuery is a high-performance, scalable, embeddable XQuery implementation that plugs easily into any Java architecture and accesses almost any data source without being dependent on underlying servers or proprietary extensions to XQuery.

For more information about Stylus XQuery, visit the following Web site:

https://www.xquery.com/

3 Analyzing EDI-to-XML Conversions

This chapter describes the EDI Analyzer API and how to use it to convert EDI to XML. It covers the following topics:

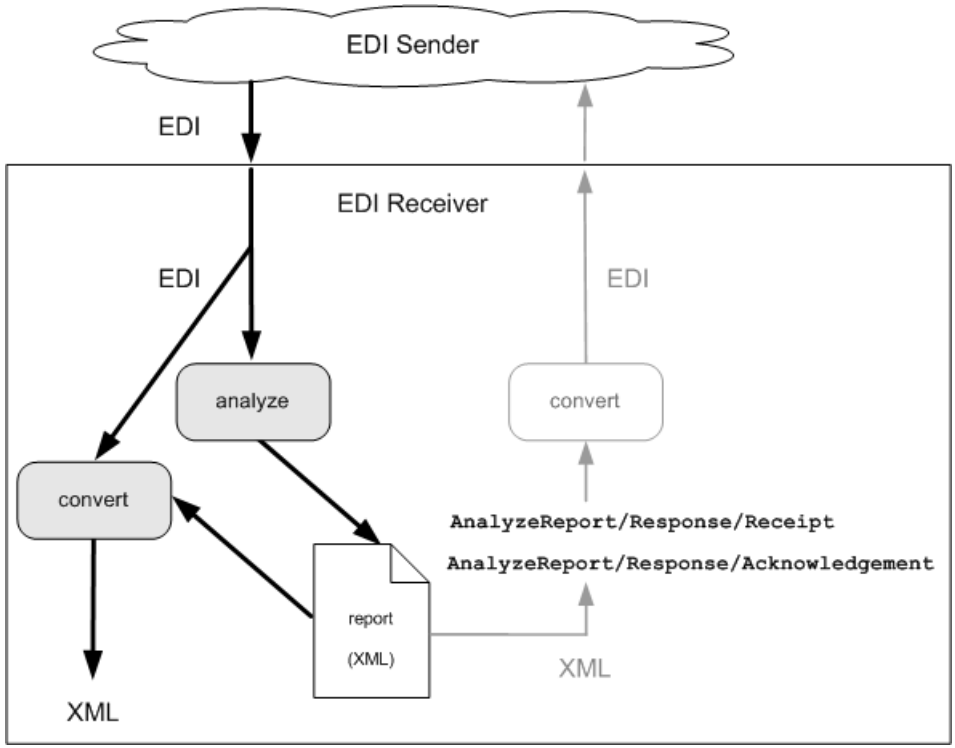
Overview

The analyze method of the EDI Analyzer API analyzes an EDI stream for warnings, errors, and fatal errors before converting the EDI stream to XML. An XML report generated by the analyze method identifies any errors in the EDI; this report is used by the convert method to filter out interchanges, groups, and messages that contain errors during conversion, allowing partial processing of EDI streams that contain errors. The analysis report also includes dialect-specific transmission response messages that can be returned to the EDI sender.

This section covers the following topics:

Illustration

The following illustration shows how to use the EDI Analyzer API to analyze and convert methods to convert EDI to XML. It shows EDI provided by an *EDI sender* (an EDI document, EDI data stored on a file system, or EDI provided by a business partner's Web service, for example) being passed to an *EDI receiver* (a separate business entity, a business partner, or an application). Transmission response messages contained in the analysis report can optionally be returned to the EDI sender.



EDI Analysis

First, the EDI data stream is analyzed by the analyze method for any errors. Errors are classified as warnings, errors, and fatal errors. Errors, along with other information about the EDI stream and its transmission, are captured in an analysis report. The analysis report is always generated by the analyze method, regardless of whether the EDI stream is free of errors. The analyze method does not throw an exception unless the input stream or output stream could not be opened.

Some input data errors can make the input data file unrecognizable to the convert method. In this case, the analysis report contains a <FatalError> element that describes the fatal error.

Analysis Report

The *analysis report* is an XML report generated by the analyze method. It identifies any interchanges, groups, or messages that contain errors and describes those errors. For some EDI dialects, the analysis report also includes a Response element, with Receipt and Acknowledgment sub-elements that you can use to send transmission responses to the EDI sender.

You can write the analysis report to any output you choose – you might want to review the report before converting the EDI to XML. The analysis report must be available to the convert method for the EDI to be converted to XML.

See “EDI Analysis Report” on page 68 for detailed information about the report’s contents and structure.

EDI Conversion

The same EDI stream specified for the analyze method must be specified for the convert method. Once the analysis is complete, you pass the analysis report to the convert method. The convert method uses the errors identified in the analysis report to filter the EDI, preventing messages containing errors from being converted to XML while allowing the rest of the messages to be converted. When using convert without the analysis report, if an error or fatal error occurs, the convert method throws an exception and no useful XML output is generated. When convert is used with the analysis report, it throws an exception in the following situations:

- If the input stream, output stream, or analysis report cannot be opened
- If the analysis report contains a <FatalError> element indicating that the input data file is unrecognizable

Specifying EDI Stream and EDI Conversion Settings

The EDI input stream specified for the analyze method must be the same as that specified in the convert method for a given XML conversion. When using a UriSource, you can use the same UriSource object for both the analyze and convert methods. If you use an InputStream source, you must rewind the input stream after calling analyze, so that convert can reread the same data.

Any conversion properties specified for the EDI stream in the analyze method must also be specified for the EDI stream in the convert method. The conversion properties are specified when the Converter object was created using the ConverterFactory.CreateConvertToXml(…) method. If you use the same Converter object for both the analyze and convert methods, you ensure that the same conversion properties are used for both.

To learn more about conversion properties, see “EDI XML Converter Properties” on page 126.

Transmission Response Messages

Some EDI specifications provide message definitions for notifying an EDI sender about the success of a transmission and about messages that were successfully processed or rejected because of errors. The analyze method automatically generates transmission response messages, creating a Response element with Receipt and Acknowledgement subelements in the EDI analysis report. Each subelement holds a complete EDI message in XML format that can be easily manipulated using XQuery and then serialized to EDI to transmit the response to the EDI sender.

Communicating wit the EDI sender is optional, and business entities have different requirements for transmitting receipt messages, acknowledgment messages, or both. See “Managing Transmission Responses” on page 76 for more information.

Dialect Support

The analyze method is supported for all EDI dialects supported by the EDI XML Converter. The analyze method is supported for EDI to XML conversions only, and not vice versa.

Automatic generation of transmission response messages is supported for the following EDI dialects:

- EDIFACT
- HIPAA
- X12

NOTES:

- EDIFACT- style messages that define a CONTRL message using EANCOM, Edig@s, and IATA (PADIS) are also supported.
- For X12 and HIPAA messages Versions 005010 and higher, you can choose whether to generate the 999 type of transaction message instead of the default 997 transaction message using the xr= URI property.

See "Managing Transmission Responses" on page 76 for more information.

Method Definition

The analyze method is defined as follows:

```
void ConvertToXML.analyze(Source source, Result result)
```

The Source and Result implementations are the same as those supported by the Convert(Source, Result) method. In the case of the analyze method, the Result object receives the EDI analysis report. Once the analysis report has been created, use the following method to convert the input data file:

```
void Convert(Source source, Result result, Source analyzeReportSource)
```

See "EDI Analysis Report" on page 68 for detailed information about the report's contents and structure. See "Example 14" on page 108 for an implementation of the analyze method in a simple Java application.

Command-Line Interface

The EDI Analyzer is supported through the command-line interface using the /analyze and /report options as part of the following command:

```
java -jar xmlconverters.jar
```

Table 3-1 provides a description of these options.

Table 3-1. EDI Analyzer Command-Line Options

Option	Description
`/analyze`	Analyzes the EDI data stream, identifying invalid items (a segment with an error, for example). Invalid items are skipped, and the rest of the data stream is converted to XML. Writes the analysis report to a temp file, which is used during the conversion, and then deleted.
`/report <Uri>`	Saves the analysis report to the specified URI. Allows later use of the transmission response messages that are automatically generated for some EDI dialects. NOTE: The /report option cannot be used alone. It must always be used with the /analyze option.

See “Command-Line Usage” on page 23 for general information about using the command-line interface.

Sending a Transmission Response

The /analyze option performs the EDI analysis and conversion as a single operation. If you want to be able to send a transmission response to the EDI sender, you must use the /report option and specify a file name to make the analysis report's Receipt and Acknowledgment elements accessible for conversion back to EDI.

See "Managing Transmission Responses" on page 76 for more information.

EDI Analysis Report

The EDI analysis report is an XML document that is generated automatically by the analyze method. It contains complete information about the transmission, including information about

- The dialect of the EDI data source
- The interchanges, groups, and messages in the transmission
- Errors
- Dialect-specific accept/reject messages

The remainder of this section describes the structure of the analysis report and provides details about the format and content of each section in the report:

- "Document Root" on page 68
- "Interchanges Element" on page 69
- "Response Element" on page 74

Document Root

The document root of the EDI analysis report is named AnalyzeReport. It has a single attribute that indicates the EDI dialect of the input document. For example:

```
<AnalyzeReport dialect="X12">
```

The Analyze root element contains two subelements: the Interchanges element and the Response element.

Interchanges Element

An *interchange* is an envelope for a set of EDI messages. The AnalyzeReport element contains a sequence of Interchange elements in which errors have been found. Interchange elements are grouped by a single Interchanges element. For example:

```
<Interchanges>
    <Interchange
       errors="false"
       firstSegment="1"
       implicit="false"
       lastSegment="11"
       sequence="1"
       warnings="true">
```

Each Interchange element defines the following attributes:

- errors – whether errors were found in the interchange
- firstSegment, lastSegment – the segment range for this interchange
- implicit – indicates if the interchange was missing, and, therefore, inferred
- sequence – the ordinal position in the input document
- warnings – whether warnings were found in the interchange

Both errors and warnings are recorded using the Error element; a severity attribute indicates the type of error: W for warning and E for error. See “Errors” on page 71 for more information about the Error element.

Segments

Each Interchange element contains a Segments element, which includes header and trailer Segment elements, as well as a Segment element for any segment containing errors.

```
<Segments>
   <Segment segnum="1" header="true" segname="ISA">
```

Each Segment element defines the following attributes:

- segnum – the absolute ordinal number for the segment in the entire EDI transmission
- header – indicates whether the segment is a header or trailer (= true) or data segment (=false)
- segname – the segment name

SegmentData

Each Segment element contains a SegmentData subelement. The specific contents of the SegmentData subelement varies based on the dialect of the EDI source being converted to XML. For an X12 EDI document, for example, the SegmentData sublement would contain an ISA element, with ISA01, ISA02, subelements, as shown here:

```
<SegmentData>
   <ISA>
     <ISA01><!--I16: Number of Included Functional-->1</ISA01>
     <ISA02><!--I12: Interchange Control Number-->32123</ISA02>
   </ISA>
</SegmentData>
```

Errors

Each Segment element can contain an Errors subelement. The Errors subelement contains one or more Error subelements. For example:

```
<Errors>
   <Error severity="E">
      <ErrorCode>DDEE0008</ErrorCode>
      <NativeErrorCode>7</NativeErrorCode>
      <NativeErrorTable>723</NativeErrorTable>
      <SegmentName>BGN</SegmentName>
      <SegmentNumber>4</SegmentNumber>
      <Value>99</Value>
      <Element>1</Element>
      <Repeat>1</Repeat>
      <Offset>3</Offset>
      <ElementName>353 (AN)</ElementName>
      <Dialect>X12</Dialect>
      <MessageVersion>004030</MessageVersion>
      <CodeListVersion>004030</CodeListVersion>
      <SystemVersion>00403</SystemVersion>
      <HeaderVersion>00403</HeaderVersion>
      <ControllingAgency>004030</ControllingAgency>
      <ErrorText>[DDEE0008] ERROR Value 99 not in codelist 353.
      Dialect: X12
      Version: 00403/004030
      Message: 831
      Segment: BGN (segment 4)
      Position: BGN01
      Element: 353 (s): Transaction Set Purpose Code
      Value: "99"
      Native error: 7, in table: 723

      The value for an element in the data stream cannot be found in the
      codelist associated with the element. Turning off codelist validation
      with "tbl=no" will eliminate the error.
      </ErrorText>
   </Error>
</Errors>
```

The Error element defines a single attribute: severity – E for error, W for warning, F for fatal.

NOTE: If the file contains the <Error severity="F"> element, it will also contain a <FatalError> element (the first child of the <AnalyzeReport> element). In this case, the analysis report cannot be used by the convert method.

Each Error element can contain the following subelements. If the element is empty, it is omitted from the report.

- ErrorCode – the vendor code
- NativeErrorCode – internal use only
- NativeErrorTable – internal use only
- SegmentName – the name of the segment in which the error occurred
- SegmentNumber – the absolute ordinal position of the segment relative to the entire EDI transmission
- Value – the field value that triggered the error
- InvalidCharacter – the invalid character that triggered the error
- Element – the absolute ordinal position of the message element where the error occurred
- Repeat – the iteration number in a loop where the error occurred
- Subelement – the particle where the error occurred
- TriElement – the particle where the error occurred
- Offset – the offset in characters from the start of the segment
- ElementName – the name of the message element where the error occurred
- Dialect – the EDI dialect name
- SyntaxVersion – the EDI syntax version

- MessageVersion – the EDI message version
- CodeListVersion – the EDI codelist version
- SystemVersion – the EDI system version
- HeaderVersion – the EDI message header version
- ControllingAgency – the agency controlling the EDI specification
- ErrorText – the complete error message

Groups

Each Interchange element also contains a Groups subelement, which contains one or more Group elements. Each Group element defines the following attributes:

- sequence – the ordinal position within the interchange
- implicit – indicates there was no group start segment, and, therefore, it was inferred
- firstSegment, lastSegment – the segment range for this group
- errors – regardless of whether errors were found in the group
- warnings – regardless of whether warnings were found in the group

Segments

Each Group element contains a Segments element. See “Segments” on page 70 for a description.

Messages

Each Group element contains a Messages element, which contains one or more Message elements. Each Message element defines the following attributes:

- sequence – the ordinal position in the input document

- implicit – indicates if the message was missing, and, therefore, inferred
- firstSegment, lastSegment – the segment range for this message
- errors – regardless of whether errors were found in the message
- warnings – regardless of whether warnings were found in the message

Response Element

Transmission responses are supported for the following types of EDI dialect messages:

- Messages that use the EDIFACT syntax and define a CONTRL message, such as EDIFACT, EANCOM, Edig@s, and IATA/PADIS
- Messages that use the X12 syntax and define a 997 or 999 message, such as X12 or HIPAA

For these EDI dialects, the analyze method generates a Response element in the EDI analysis report. The Response element contains Receipt and Acknowledgement subelements:

```
<Response>
   <Receipt>...</Receipt>
   <Acknowledgement>...</Acknowledgement>
</Response>
```

The Receipt and Acknowledgement elements each contain a complete EDI message in XML format. Receipt element messages indicate only that the transmission from the EDI sender was received; they contain no information about the content of the transmission. Acknowledgement element messages contain information for each interchange, group, and message that was received, including whether it was accepted without errors, rejected, or partially accepted.

Receipt and Acknowledgement elements can be easily manipulated using XQuery and then serialized to EDI for consumption by the EDI sender.

For EDI dialects for which transmission responses are not supported, the analysis report contains an empty Response element.

See “Managing Transmission Responses” on page 76 for more information.

Managing Transmission Responses

Some EDI specifications define the interchanges and messages to be used to notify the EDI sender about transmission status, from initial receipt of the transmission to the errors, if any, encountered in the EDI data stream received from the EDI sender. For example:

- HIPAA and X12 use the TA1 interchange to indicate whether a transmission was accepted, accepted with errors, or rejected. The 997 or 999 transaction sets, as set by the xr=URI property, are used to report errors encountered during EDI processing.
- EDIFACT uses the CONTRL message to indicate acceptance or rejection of a transmission, and also to report errors encountered during EDI processing.

  NOTE: This also includes EDI dialects that use the EDIFACT syntax such as EANCOM, Edig@s, and IATA/PADIS.

As described in “Response Element” on page 74, the analyze method generates dialect-specific transmission responses as Receipt and Acknowledgement elements in the EDI analysis report.

This section covers the following topics:

Receipt Element Example

Here is an example of the Receipt element from the analysis report created using the analyze method to convert the sample file threemsgs.x12 to XML:

```
<Receipt>
    <X12>
        <ISA>
            <ISA01>00</ISA01>
            <ISA03>00</ISA03>
            <ISA05>01</ISA05>
            <ISA06>5151515151</ISA06>
            <ISA07>01</ISA07>
            <ISA08>1515151515</ISA08>
            <ISA11>^</ISA11>
            <ISA12>00403</ISA12>
            <ISA13>0</ISA13>
            <ISA14>0</ISA14>
            <ISA15>P</ISA15>
            <ISA16>*</ISA16>
        </ISA>
        <TA1>
            <TA101>32123</TA101>
            <TA102>041201</TA102>
            <TA103>1217</TA103>
            <TA104>A</TA104>
            <TA105>000</TA105>
        </TA1>
        <IEA/>
    </X12>
</Receipt>
```

The specific structure of the Receipt element varies based on the dialect, and contents, of the EDI stream being converted. In this example:

- The X12 element specifies the EDI dialect of the EDI data stream that was converted to XML.
- The ISA element represents the Interchange Control Header segment; its subelements (ISA01, ISA02, and so on) show the values for the corresponding segment fields (Authorization Information Qualifier, Authorization Information, and so on).
- The TA1 element represents the Transaction Acknowledgement segment; its subelements (TA101, TA102, and so on) show the values for the corresponding segment fields (Interchange Control Number, Interchange Date, and so on).
- The IEA element represents the Interchange Control Trailer segment; it is empty because the values for this segment are computed automatically by Stylus XML Converters when the Receipt element is converted to EDI for transmission back to the EDI sender.

  Other EDI segments that are computed when the XML is converted to EDI include the Transaction Set Trailer (SE) and Function Group Trailer (GE).

Acknowledgement Element Example

Following is an example of the Acknowledgement element from the same analysis report created using the analyze method to convert the sample file threemsgs.x12 to XML. Note that is has been abbreviated for formatting considerations.

```
<Acknowledgement>
    <X12>
        <ISA> … </ISA>
        <GS> … </GS>
        <TS_997>
            <ST>
                <ST01>997</ST01>
                <ST02>0</ST02>
            </ST>
            <AK1>
                <AK101>CT</AK101>
                <AK102>128</AK102>
            </AK1>
            <AK2>
                <AK201>831</AK201>
                <AK202>00128001</AK202>
            </AK2>
            <AK5> … </AK5>
            <AK2>
                <AK201>831</AK201>
                <AK202>00128002</AK202>
            </AK2>
            <AK3> … </AK3>
            <AK4>
                <AK401> … </AK401>
                <AK402>782</AK402>
                <AK403>6</AK403>
                <AK404>ZZZZ</AK404>
            </AK4>
            <AK5> … </AK5>
            <AK2> … </AK2>
                <AK201>831</AK201>
                <AK202>00128003</AK202>
```

```
                    <AK5> … </AK5>
                    <AK9>
                        <AK901>P</AK901>
                        <AK902>3</AK902>
                        <AK903>3</AK903>
                        <AK904>2</AK904>
                    </AK9>
                    <SE/>
                </TS_997>
                <GE/>
                <IEA/>
            </X12>
        </Acknowledgement>
```

In this example:

- The X12 and ISA elements serve the same function as those in the Receipt element.
- The GS element represents the Functional Group Header segment; its subelements (GS01, GS02, and so on) show the values for the corresponding segment fields (Functional Identifier Code, Application Sender's Code, and so on).
- The TS_997 element serves as a message wrapper for transaction messages. The TS represents *transaction set*, and 997 indicates the type of message.

  NOTES:

  - Acknowledgement transaction message wrapper elements have different names in different EDI dialects. For example, in EDIFACT-style messages, the acknowledgment transaction message uses the CONTRL message.
  - The type of transaction message that is generated for X12 (997 or 999) depends on the xr= URI property. See Table 5-9, "Properties for the EDI XML Converter," on page 127 for details about setting the xr= URI property. The preceding example shows a functional acknowledgement transaction message as indicated by the TS_997 element. Similarly, the

TS_999 element is the message wrapper for X12 implementation acknowledgement transaction messages.

- The ST element represents the Transaction Set Header.
- The AK elements represent the following items:
  - Functional Group Response Header (AK1) and Functional Group Response Trailer (AK9) segments. There is one pair of AK1/AK9 segments for each group of transactions.
  - Transaction Set Response Header (AK2) and Transaction Set Response Trailer (AK5). These pairs of segments can repeat once for each transaction in the transaction set. In this example, there are three AK2 segments because there are three messages in the threemsgs.x12 EDI source document.

  If a message contains an error, the analysis report can also contain elements representing the following segments:
  - AK3 (Data Segment Note). This segment identifies the invalid segment's position within the transaction, as well as an error code that specifies the type of error.
  - AK4 (Data Element Note). If the segment is determined to be invalid because of bad data (a value with an improper data type, for example), the AK4 subelements specify the Data Element Syntax Error Code (AK403) and Copy of Bad Data Element (AK404).
- The SE, GE, and IEA segments are the same as those described in "Receipt Element Example" on page 77.

NOTE: The Acknowledgement element can also include a TA1 element, as controlled by a combination of the ISA14 element in the incoming transaction set and the ta1= URI property. If a TA1 element is generated, it appears immediately before the ISA element.

Converting Response Messages to EDI

Because transmission responses are structured as XML, they need to be converted to EDI before they can be returned to the EDI sender. This example takes you through the following tasks:

- Use the analyze method to generate the analysis report and convert the EDI data stream (in this case, a sample X12 EDI document, threemsgs.x12) to XML.
- Locate the Receipt element in the analysis report.
- Convert the XML for the TA1 Transaction Acknowledgement segment to EDI.

To see a complete example application that converts both Receipt and Acknowledgement responses to EDI, see “Example 14” on page 108.

Invoking the analyze Method

The EDI XML Converter is used to initiate the conversion of the source EDI document, threemsgs.x12 to EDI:

```
InputStream inStream = null;
   try {
   Source ediSource = new StreamSource(exampleURI + "threemsgs.x12");
   ConvertToXML toXml = factory.newConvertToXML("converter:EDI");
```

Next, the analyze method is used to generate the analysis report and save the output to report.xml:

```
Result reportResult = new StreamResult("report.xml");
toXml.analyze(ediSource, reportResult);
```

See “Receipt Element Example” on page 77 for a sample of the analysis report, report.xml.

Converting the Source EDI

The analysis report is used as input to convert the EDI stream to XML. Any errors in the EDI stream are recorded in the analysis report, which is used by ConvertToXML as a filter so that only valid EDI messages are converted to XML. Here, the valid EDI is written to an XML document, twomsgs.xml.

```
Source reportSource = new StreamSource("report.xml");
Result xmlResult = new StreamResult("twomsgs.xml");
toXml.convert(ediSource, xmlResult, reportSource);
```

Locating Response Messages

The EDI analysis report is used again, this time as the source for the EDI transmission response messages that have been generated in the Receipt and Acknowledgement elements in the XML report. The content of these elements are converted to EDI for transmission back to the EDI sender. Here, the report is opened with an instance of XMLStreamReader.

```
inStream = new FileInputStream("report.xml");
XMLStreamReader rdr =
   XMLInputFactory.newInstance().createXMLStreamReader(inStream);
```

Once the XMLStreamReader object is created, we can read through the analysis report, skipping first to the Receipt element, then to the X12 element:

```
do {
   rdr.next();
   } while(rdr.getEventType() != XMLStreamConstants.START_ELEMENT
      || !rdr.getName().getLocalPart().equals("Receipt"));
      do {
         rdr.next();
      } while(rdr.getEventType() != XMLStreamConstants.START_ELEMENT
         || !rdr.getName().getLocalPart().equals("X12"));
```

For a refresher of the Receipt element structure, see “Receipt Element Example” on page 77.

Converting the Receipt Element to EDI

Once the Receipt element is located in the analysis report, it can be converted to EDI for transmission back to the EDI sender. Note that the converter: property is specified as EDI.

```
ConvertFromXML converter = factory.newConvertFromXML("converter:EDI");
Source responseSource = new XMLStreamReaderSource(rdr);
Result receiptResult = new StreamResult("receipt.x12");
converter.convert(responseSource, receiptResult);
```

The resulting EDI is written to the file, receipt.x12, which contains the following:

```
ISA+00+          +00+          +01+5151515151
+01+1515151515     +090818+1110+^+00403+000000000+0+P+*'
TA1+000032123+041201+1217+A+000'
IEA+0+000000000'
```

Notice that the IEA segment (Interchange Control Trailer), which was represented in the original XML conversion of the source EDI document as an empty element (<IEA/) has been automatically computed by the EDI XML Converter and now includes values for the Number of Included Functional Groups (IEA01) and Interchange Control Number (IEA02) segments.

Sending Responses to the EDI Sender

Typically, when you send a response to an EDI sender, you must provide each interchange with a unique identifier. For X12, for example, this is the Interchange Control Number (ISA12) segment; for EDIFACT, this is the Interchange Control Reference (UNB05) segment.

You can perform this task as part of the application code that extracts the Receipt or Acknowledgement element from the analysis report; it is expected that the generation of a unique identifier is handled elsewhere.

4 Stylus XML Converters® Examples

The Stylus XML Converters® API allows you to access non-XML files and convert them to XML, and vice versa. The converter: URIs used to access data sources can be invoked programmatically, in an XQuery application, for example. This facility allows you to treat non-XML data as XML, manipulate it as needed, and, optionally, write it back to its source in its original format.

This chapter describes demo.java, a simple Java program installed in the Stylus XML Converters \examples directory that demonstrates some of the features of Stylus XML Converters and the Converters API.

This chapter also provides examples of using Stylus XML Converters that are not part of the \examples directory.

Overview of the demo.java Example

The example file, demo.java, runs several sample demonstrations that show how the XML Converters API can be used to convert data to and from XML stored in a number of different formats using both Stylus XML Converters and user-defined custom XML conversions created using Stylus Studio. This section describes the files associated with the demonstrations and how to run it.

Examples Summary

The examples included in demo.java are summarized in the following table.

Example	Description
Example 1	Show s a simple conversion of a comma-separated values (CSV) file to XML.
Example 2	Shows how to convert an XML file to CSV.
Example 3	Shows how to use a custom XML converter to convert a fixed-width file to XML.
Example 4	Shows how to perform a conversion to XML and then transform the result.
Example 5	Shows how to use the result of a transformation as input to an XML conversion.
Example 6	Simple example showing how to use the EDI XML Converter.
Example 7	Shows how to use a Standard Exchange Format (SEF) extension file to convert EDI messages using a proprietary format.
Example 8	Shows how to use the `ConverterListener` to manage warnings and errors during the conversion process.
Example 9	Shows how to generate an XML Schema from a CSV file.
Example 10	Shows how to generate an XML Schema from an EDI file.
Example 11	Shows how to use the XPath document() function in an XSLT stylesheet object to take a converter: URI as its argument.
Example 12	Shows how to convert an EDI file into an org.w3c.dom interface node contained entirely in memory.
Example 13	Shows how to convert an EDI file into an XMLStreamReader, which can then be used to read the XML data entirely from memory in a streaming fashion.
Example 14	Shows how to use the analyze() method to analyze an EDI stream for errors as part of the conversion process.

Demonstration Files

The files required to run the demonstrations are summarized in the following table. The files are installed in the *<install dir>*\examples directory, where installdir is your product installation directory.

File	Description
831.x12	EDI file used in Example 6.
copier.xslt	XSLT used in Example 4 and Example 5.
demo.bat	The demonstration driver batch file.
demo.class	The demonstration class file.
demo.java	The source for the demonstration; this file contains the usage comments.
demo.sh	The demonstration driver file for UNIX.
one.csv	The input file for the first example run by demo.bat.
proprietary.sef	The SEF file that defines non-standard X12 message types; used in Example 7.
proprietary.x12	Sample X12 with a non-standard message type; used in Example 7.
three.conv	The definition for the custom XML conversion used in the third example run by demo.bat.
three.txt	The input file for the third example run by demo.bat.
threemsgs.x12	The EDI input file used for Example 14.
two.xml	The input file for the second example run by demo.bat.

Running demo.java

This section describes the requirements and procedure for running the demonstration application, demo.java.

Before You Begin

To recompile the files in the demonstration application, ensure that Java 1.8 or later is installed on your machine, and that its \bin directory is included in your system's PATH.

Running the Demonstration

1. Open a console window.
2. Change to the `<install dir>\examples` directory.
3. Run demo.bat (or demo.sh, if you installed Stylus XML Converters on a UNIX/Linux machine).

Example 1

Example 1 converts a comma-separated values (CSV) file, one.csv, to an XML file, one.xml, using the CSV XML Converter. The conversion parameter for the new Converter object is specified as a converter: URL that indicates which XML Converter to use to convert the input file to the output file. Only two XML Converter property settings are expressed; default values are used for all properties unless you specify them in the converter: URL.

```
try {
   Source converterSource = new StreamSource(exampleDir + "one.csv");
   Result converterResult = new StreamResult("one.xml");

   ConvertToXML toXml = factory.newConvertToXML("converter:CSV:sep=,:first=yes");
   toXml.convert(converterSource, converterResult);

   System.out.println("test 1 finished: one.csv -> one.xml");
   }
catch(Exception e) {
   System.out.println("test 1 failed with exception: " + e);
   }
```

Both input and output streams are opened and closed by the Converter object.

Example 2

Example 2 is similar to Example 1, but instead of converting a non-XML file to XML, it does the opposite. It also shows how to use the URI resolver to create both the input stream and output stream:

```
StreamSource converterSource = null;
FileOutputStream streamResult = null;
   try {
      ConverterResolver resolver = factory.newResolver();
      converterSource = (StreamSource)resolver.resolve("two.xml", exampleDir);
      streamResult = new FileOutputStream("two.csv");
      Result converterResult = new StreamResult(streamResult);
```

In this example, we need to close the input and output streams because we opened them, not the Converter object.

Example 3

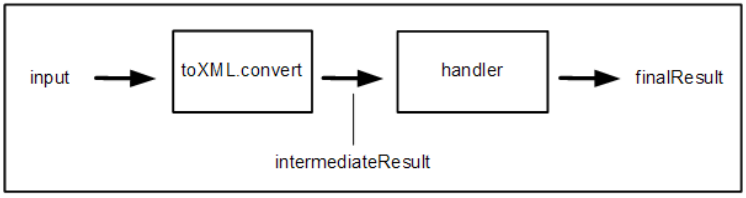
Example 3 uses a custom XML conversion, three.conv, built using Stylus Studio XML Enterprise Suite, to convert a fixed-width file, three.txt, to XML. Here, we create our own Streams. Because we are converting a local text file, there is no need to use the URI Resolver.

```
try {
   Source converterSource = new StreamSource(exampleDir + "three.txt");
   Result converterResult = new StreamResult("three.xml");

   String customConversion = "converter:" + exampleDir + "three.conv";

   Converter toXml = factory.newConvertToXML(customConversion);
   toXml.convert(converterSource, converterResult);

   System.out.println("test 3 finished: three.txt -> three.xml");
   }
catch(Exception e) {
   System.out.println("test 3 failed with exception: " + e);
   }
```

Example 4

Examples 1, 2, and 3 performed a simple conversion of one file type to another – some type of converter (either Stylus XML Converters or a user-defined custom XML conversion) was given an input and converted it to another format.

Example 4 generates conversion output as SAX events (in this case, a CSV file, one.csv, is converted to XML using the Stylus XML Converters). These SAX events are then sent to the transformer's ContentHandler. The process is summarized in the following illustration.



Here is the code for Example 4:

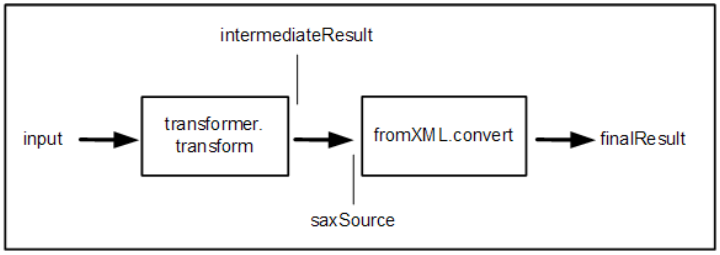
```
try {
   Source converterSource = new StreamSource(exampleDir + "one.csv");

   ConvertToXML toXml = factory.newConvertToXML("converter:///CSV:sep=,
      :first=yes");

   SAXSource converterResult = toXml.getSAXSource(converterSource);

   StreamSource xsltSource = new StreamSource(exampleDir + "copier.xslt");
   StreamResult xsltResult = new StreamResult("four.xml");

   TransformerFactory transformerFactory = TransformerFactory.newInstance();
   Transformer xslt = transformerFactory.newTransformer(xsltSource);

   xslt.transform(converterResult, xsltResult);

   System.out.println("test 4 finished: one.csv -> four.xml");
   }
catch(Exception e) {
```

```
System.out.println("test 4 failed with exception: " + e);
e.printStackTrace(System.out);
}
```

XMLWriter StreamResult is an XML document, four.xml. In this example, we used a copy/identity transformation, but you could specify any XSLT transformation here to perform any processing on the intermediate result you required.

Example 5

In Example 5, output from an XSLT transformation is sent to a converter, which takes the XML that is written to it (as a saxSource) and converts it to CSV. This process is summarized in the following illustration.



Here is the code for Example 5:

```
try {
   StreamResult converterResult = new StreamResult("five.csv");
   SAXSource converterSource = new SAXSource();

   Converter fromXml = factory.newConvertFromXML("converter:CSV:sep=,
      :first=yes");
   fromXml.convert(converterSource, converterResult);

   SAXResult xsltResult = new SAXResult();
   xsltResult.setHandler(converterSource.getXMLReader().getContentHandler());

   StreamSource xsltSource = new StreamSource(exampleDir + "copier.xslt");
   StreamSource xsltInput = new StreamSource(exampleDir + "two.xml");

   TransformerFactory transformerFactory = TransformerFactory.newInstance();
   Transformer xslt = transformerFactory.newTransformer(xsltSource);
   xslt.transform(xsltInput, xsltResult);
   System.out.println("test 5 finished: two.xml -> five.csv");
   }
catch(Exception e) {
   System.out.println("test 5 failed with exception: " + e);
   }
```

To convert the transformation's output to CSV, we have used an instance of the ConvertFromXML object. This object uses the XML Converters CSV converter.

Example 6

Example 6 shows the use of an EDI XML Converter (converter: EDI) to convert a file in the X12 dialect (831.x12) to XML (831.x12.xml), and then back to EDI (831.x12.xml.fromxml).

```
try{
   Source converterSource = new StreamSource(exampleDir + "831.x12");
   Result converterResult = new StreamResult("831.x12.xml");
   Converter toXml = factory.newConvertToXML("converter:EDI");
   toXml.convert(converterSource, converterResult);
   System.out.println("test6 toXML finished: 831.x12 -> 831.x12.xml");
   }
catch (Exception e)
   {
   System.out.println("test 6 toXML failed with exception: " + e);
   }

try{
   Source converterSource = new StreamSource("831.x12.xml");
   Result converterResult = new StreamResult("831.x12.fromxml");
   Converter fromXml = factory.newConvertFromXML("converter:EDI");
   fromXml.convert(converterSource, converterResult);

System.out.println("test6 fromXML finished: 831.x12.xml -> 831.x12.fromxml");
   }
catch (Exception e)
   {
   System.out.println("test 6 fromXML failed with exception: " + e);
   }
```

Example 7

This example shows how to use a Standard Exchange Format (SEF) extension file to convert EDI messages using a proprietary format. The SEF file used in this example adds 99 as a code value in the 353 element of segment BGN in transaction set 831.

The URL of the SEF file is specified in the user= property of the converter: URI. You can specify the SEF file name as a relative path. For example, if Stylus XML Converters are installed in a directory named XML_CONVERTERS and the EDI converter: URI contains `user=relative.sef`, the SEF file can be found at `XML_CONVERTERS/lib/CustomEDI/relative.sef`.

```
try{
   String sefUrl = new File(exampleDir
      +"proprietary.sef").toURI().toString();
   String ediUrl = "converter:EDI:user=" + sefUrl;
   Source converterSource = new StreamSource(exampleDir + "proprietary.x12");
   Result converterResult = new StreamResult("proprietary.xml");
   Converter toXml = factory.newConvertToXML(ediUrl);
   toXml.convert(converterSource, converterResult);
   System.out.println("test 7 toXML finished: proprietary.x12 ->
      proprietary.xml");
   }
catch (Exception e)
   {
   System.out.println("test 7 toXML failed with exception: " + e);
   }
```

Example 8

This example shows how to register a `ConverterListener`, which is notified of warnings, errors, and fatal errors that occur during conversion. Also included in this example is a simple implementation of the three `ConverterListener` methods (`warning`, `error`, and `fatalError`). This implementation appears at the end of demo.java.

This example uses the proprietary data file (proprietary.xml) as "Example 7" on page 98, but it omits the proprietary.sef extension file. This will result in a error that is reported to the `ConverterListener` implementation.

Sample Application

```
try{
   Source converterSource = new StreamSource(exampleDir + "proprietary.x12");
   Result converterResult = new StreamResult("proprietary.xml");
   Converter toXml = factory.newConvertToXML("converter:EDI");

   ConverterListener listener = new DemoListener();
   toXml.getConfiguration().setConverterListener(listener);
   toXml.convert(converterSource, converterResult);
   System.out.println("test 8 toXML finished: proprietary.x12 ->
      proprietary.xml");
   }
catch (Exception e)
   {
   System.out.println("test 8 toXML failed with exception: " + e);
   }
```

ConverterListener Implementation in demo.java

```
public static class DemoListener implements ConverterListener {

   public void warning(ConverterException e) throws ConverterException {
      System.out.println("Converter warning notification: " + e);
      return;
   }

      public void error(ConverterException e) throws ConverterException {
      System.out.println("Converter error notification: " + e);
      return;
      }

   public void fatalError(ConverterException e) throws ConverterException {
      System.out.println("Converter fatal error notification: " + e);
      return;
   }
}
```

Error Listener Output

After running demo.java, the program generates the following output. Notice the error that is encountered after completing Example 7.

```
test 1 finished: one.csv -> one.xml
test 2 finished: two.xml -> two.csv
test 3 finished: three.txt -> three.xml
test 4 finished: one.csv -> four.xml
test 5 finished: two.xml -> five.csv
test 6 toXML finished: 831.x12 -> 831.x12.xml
test 6 fromXML finished: 831.x12.xml -> 831.x12.fromxml
test 7 toXML finished: proprietary.x12 -> proprietary.xml
```

```
Starting test 8.  The ConverterListener will print a warning and an error.

Converter warning notification:
com.ddtek.xmlconverter.adapter.edi.EDIConverterException: [DDEW0063] WARNING
Starting with 00402, the format of ISA11 changed. Attempting to adjust 'U' to
'^'.
   Segment: ISA (segment 1)

In X12 prior to 004020, ISA11 was element I10 and had to have the value "U".
But from 004020 onwards, it
is element I65 and is the repetition character. The default automatic fixups
will mske this change to the
data stream unless explicitly disabled.
Converter error notification:
com.ddtek.xmlconverter.adapter.edi.EDIConverterException: [DDEE0008] ERROR
Value 99 not in codelist 353.
   Dialect: X12
   Version: 00403/004030
   Message: 831
   Segment: BGN (segment 4)
   Position: BGN01
   Element: 353 (s): Transaction Set Purpose Code
   Value: "99"
   Native error: 7, in table: 723

The value for an element in the data stream cannot
be found in the codelist associated with the element.
Turning off codelist validation with "tbl=no" will
eliminate the error.
test 8 toXML finished: proprietary.x12 -> error.xml

test 9 schema generator finished: one.csv -> one.xsd
test 10 schema generator finished: --> edi.xsd
test 11 finished: one.xml + one.csv -> eleven.xml
test 12 finished: 831.x12 --> DOM in memory.org.
test 13 finished: 831.x12 --> XMLStreamReader containing 220 events.
test 14 finished: threemsgs.x12 --> twomsgs.xml, receipt.x12,
acknowledgement.x12
```

Example 9

This example shows how to use the Converter API to create an XML Schema based on a comma-separated values (CSV) file.

NOTE: An instance document is required to generate an XML Schema for CSV and other file types. See “Instance Documents” on page 33 for more information.

The XML Schema generator is used in the same way as an XML Converter – the program provides the sample input file as a Source object and the generated XML Schema is written to the Result object.

See “XML Schema Generation” on page 30 for more information on this topic.

```
try{
   Source sampleSource = new StreamSource(exampleDir + "one.csv");
   Result xsdResult    = new StreamResult("one.xsd");
   SchemaGenerator generator =
      factory.newSchemaGenerator("converter:///CSV:sep=,:first=yes");
   generator.getSchema(sampleSource, xsdResult);
   System.out.println("test 9 schema generator finished: one.csv -> one.xsd");
   }
catch (Exception e)
   {
   System.out.println("test 9 schema generator failed with exception: " + e);
   }
```

Example 10

This example shows how to use the Converter API to create an XML Schema based on an EDI file. The generated XML Schema depends on the EDI dialect, version, and message being converted, but not on the actual data in the EDI message. This information can be provided in the following ways:

- Using a sample EDI input (as shown in “Example 9” on page 102). See “Instance Documents” on page 33 for more information.
- As part of the converter: URI, as demonstrated in this example. Also, see “Converter URI Properties” on page 33 for more information.

See “XML Schema Generation” on page 30 for more information.

```
try{
   Result xsdResult    = new StreamResult("edi.xsd");
   String uri = "EDI:dialect=EDIFACT:version=D06B:message=INVOIC:tbl=no";
   SchemaGenerator generator = factory.newSchemaGenerator(uri);
   generator.getSchema(xsdResult);
   System.out.println("test 10 schema generator finished: --> edi.xsd");
   }
catch (Exception e)
   {
   System.out.println("test 10 schema generator failed with exception: " + e);
   }
```

Example 11

This example shows how to use a document URI resolver (com.ddtek.xmlconverter.ConverterResolver, the XML Converter implementation of javax.xml.transform.URIResolver) to enable the XPath document() function in an XSLTstylesheet object to take a converter: URI as its argument.

The first statement uses newResolver() to get the URI resolver, which is able to resolve converter: URIs for the document() function.

Next, the example creates a converter: URI like this:

```
converter:///CSV:sep=,:first=yes?file:///c:/examples/one.csv
```

The XML Converter reads its input from the one.csv file, converts it to XML, and returns its document node to the document() function.

Previous examples used a converter: URI like this:

```
converter:///CSV:sep=,:first=yes
```

and the name of the input file was provided elsewhere. When using the document() function, you must provide the converter: URI and the input file URI, separating them with a question mark (?) as shown in this example.

Here is the complete code for Example 11:

```
try {

   URIResolver resolver = factory.newResolver();
   String converterUri = "converter:///CSV:sep=,:first=yes?" + exampleDir +
                         "one.csv";

   String xsltString =
      "<xsl:stylesheet version='1.0'
         xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>"
        + "  <xsl:output method='xml' indent='yes'/>"
        + "  <xsl:template match='/'>"
         + "    <root>"
         + "        <xsl:copy-of select='.'/>"
```

```
         + "        <xsl:copy-of select='document(\"" + converterUri +
"\")'/>"
         + "     </root>"
         + "   </xsl:template>"
         + " </xsl:stylesheet>";

   StreamSource xsltSource = new StreamSource(new StringReader(xsltString));
   Transformer xslt =
      TransformerFactory.newInstance().newTransformer(xsltSource);
   xslt.setURIResolver(resolver);

   StreamSource xmlSource = new StreamSource(exampleDir + "one.xml");
   StreamResult transformedResult= new StreamResult(exampleDir +
      "eleven.xml");
   xslt.transform(xmlSource, transformedResult);
   System.out.println("test 11 finished: one.xml + one.csv -> eleven.xml");
   }
   catch (Exception e) {
      System.out.println("test 11 failed with exception: " + e);
   }
}
}
```

Example 12

This example shows how to use XML Converters™ to convert an EDI file into an org.w3c.dom interface node contained entirely in memory. In the example, the toXml.convert call performs the conversion and leaves the resulting DOM in the converterResult object.

```
try{
   Source converterSource = new StreamSource(exampleURI + "831.x12");
   DOMResult converterResult = new DOMResult();
   Converter toXml = factory.newConvertToXML("converter:EDI");
   toXml.convert(converterSource, converterResult);
   Node resultDocument = converterResult.getNode();

   System.out.println("test 12 finished: 831.x12 --> DOM in memory.org.");
   }
   catch (Exception e) {
      System.out.println("test 12 failed with exception: " + e);
   }
```

Example 13

This example shows how to use Stylus XML Converters to convert an EDI file into an XMLStreamReader, which can then be used to read the XML data. XMLStreamReader processes data entirely in memory in a streaming fashion, allowing efficient processing of input files of unlimited size.

Notice that there is no call to convert(...) in this example. The getXMLStreamReader call is a convenience method that performs the conversion and returns the XMLStreamReader in one call. The example then reads and counts all the parsing events from the XMLStreamReader.

In a real application, the program would process those events as they are read.

```
try{
   Source converterSource = new StreamSource(exampleURI + "831.x12");
   ConvertToXML toXml = factory.newConvertToXML("converter:EDI");
   XMLStreamReader rdr = toXml.getXMLStreamReader(converterSource);
   int eventCount = 0;
   while(rdr.hasNext()) {
      rdr.next();
      eventCount++;
   }

   System.out.println("test 13 finished: 831.x12 -->
      XMLStreamReader containing " + eventCount + " events.");
   }
   catch (Exception e) {
      System.out.println("test 13 failed with exception: " + e);
   }
   }
}
```

Example 14

This example shows how to use the EDI Analyzer API to convert an input EDI document, threemsgs.x12, to XML. The source EDI contains three messages, one of which contains an error. This example shows how to use the EDI analysis report generated by the analyze() method to filter the invalid message from the EDI while converting the rest of the EDI stream to XML. Finally, it shows how to convert the EDI analysis report's Receipt and Acknowledgement elements to EDI for transmission back to the EDI sender.

For more information about using the EDI Analyzer API, see Chapter 3, "Analyzing EDI-to-XML Conversions."

```
InputStream inStream = null;
try {
   Source ediSource = new StreamSource(exampleURI + "threemsgs.x12");
   ConvertToXML toXml = factory.newConvertToXML("converter:EDI");

   Result reportResult = new StreamResult("report.xml");
   toXml.analyze(ediSource, reportResult);

   Source reportSource = new StreamSource("report.xml");
   Result xmlResult = new StreamResult("twomsgs.xml");
   toXml.convert(ediSource, xmlResult, reportSource);

   inStream = new FileInputStream("report.xml");
   XMLStreamReader rdr = XMLInputFactory.newInstance()
      .createXMLStreamReader(inStream);

   do {
      rdr.next();
   } while( rdr.getEventType() != XMLStreamConstants.START_ELEMENT
      || !rdr.getName().getLocalPart().equals("Receipt"));

      do {
         rdr.next();
      } while( rdr.getEventType() != XMLStreamConstants.START_ELEMENT
         || !rdr.getName().getLocalPart().equals("X12"));
```

```
      ConvertFromXML converter = factory.newConvertFromXML ("converter:EDI");
      Source responseSource = new XMLStreamReaderSource(rdr);
      Result receiptResult = new StreamResult("receipt.x12");
      converter.convert(responseSource, receiptResult);

      do {
         rdr.next();
      } while( rdr.getEventType() != XMLStreamConstants.START_ELEMENT
         || !rdr.getName().getLocalPart().equals("Acknowledgement"));

      do {
         rdr.next();
      } while( rdr.getEventType() != XMLStreamConstants.START_ELEMENT
         || !rdr.getName().getLocalPart().equals("X12"));


         Result ackResult = new StreamResult("acknowledgement.x12");
         converter.convert(responseSource, ackResult);

         System.out.println("test 14 finished: threemsgs.x12 -->
                             twomsgs.xml, receipt.x12, acknowledgement.x12");

      catch (Exception e) {
         System.out.println("test 14 failed with exception: " + e);
      }
      finally {
         if (inStream != null)
         try {inStream.close();} catch(IOException e) {}
      }
}
```

Processing Conversion Results

You can use the OutputResult class to write a Converter's output to a stream. Using this implementation, conversion results are written as a whole (as an XML document, for example) once the conversion is complete. This technique is known as *pushing* results.

Alternatively, you can treat conversion results one-at-a-time as XML fragments instead of an entire XML document. This technique is known as *pulling* results, and it can be accomplished using the public XMLStreamReader interface, as shown in the following example:

```
try{
   Source converterSource = new StreamSource(exampleURI + "831.x12");
   ConvertToXML toXml = factory.newConvertToXML("converter:EDI");
   XMLStreamReader rdr = toXml.getXMLStreamReader(converterSource);
   int eventCount = 0;
   while(rdr.hasNext()) {
      rdr.next();
      eventCount++;
   }

   System.out.println(
   "test 13 finished: 831.x12 --> XMLStreamReader containing " + eventCount
   + " events.");

   catch (Exception e) {
      System.out.println("test 13 failed with exception: " + e);
   }
}
```

Loading SEF Files Programmatically

The setEDIExtension() method allows you to reference a SEF file programmatically. This method is a member of the Configuration class. It is defined as:

```
void setEDIExtension(javax.xml.transform.stream.StreamSource source)
   throws ConverterException
```

Because a SEF file can be used by the XML Schema generator, the following instance method was added to the SchemaGenerator class to provide access to the Configuration object:

```
Configuration getConfiguration()
```

Stylus XML Converters reads the source object and parses the data as a SEF file, creating an Extender object. All XML Converters created using that Configuration object use that Extender object as if it had been supplied with the user= option in the URI.

Stylus XML Converters uses a byte stream if the source object contains one. If the source object does not contain a byte stream, Stylus XML Converters uses a character stream, if present. If neither a byte stream nor a character stream is available, Stylus XML Converters uses a systemId. If none of these is present in the source object, Stylus XML Converters throws an exception.

Using SEF Files Created with Stylus Studio

In addition to custom segment and message definitions, SEF files created using the Stylus Studio EDI to XML Module can contain a converter: URI in a section called .PRIVATE. This converter: URI can contain properties (val=no and len=yes, for example).

If you specify such a SEF file in your application, Stylus XML Converters uses the converter properties from that URI when performing the XML conversion. That is, the values specified for the properties in the SEF become the new default values for the properties. This behavior is also true when the SEF file is loaded with the user= URI property.

Using a SEF File for Multiple Conversions

The Configuration object owns the Extender. If you want to use a SEF file for multiple conversions, you can do so as follows:

```
ConverterFactory factory = new ConverterFactory();
factory.getConfiguration().setEDIExtension (StreamSource sef);
```

All Converter and SchemaGenerator objects created from that factory have access to the loaded SEF file, but they do not parse the SEF file each time they are created.

If setEDIExtension is called two times, then the first SEF file is replaced by the second one. Any Converter or SchemaGenerator objects already created will still use the first SEF file.

If you want to use a SEF file for one Converter or SchemaGenerator object only, you can do so as follows:

```
ConverterFactory factory = new ConverterFactory();
Converter converter = factory.newConvertToXML(…);
Converter.getConfiguration().setEDIExtension (StreamSource sef);
```

5 Stylus XML Converters® Properties

Stylus XML Converters share some properties (the line separator property, for example), and each has properties that are unique. For example, the CSV XML Converter allows you to specify an escape character, but the binary XML Converter does not.

This chapter contains reference information for the Stylus XML Converters including information about the line separator property and encoding values, which are common to most Stylus XML Converters.

Line Separator Values

Most Stylus XML Converters allow you to specify a *line separator* character, a character that signifies the end of a line of text. The line separator character is specified in the converter URI using the newline= property.

The following table provides a list of commonly used values. All values are case-insensitive.

Table 5-1. Line Separator Values

Value	Result (Hexadecimal)
platform	The default from System.getProperty("line.separator")
cr	CR (0D)
lf	LF (0A)
unix	LF (0A)
crlf	CR, LF (0D, 0A)
dos	CR, LF (0D, 0A)
windows	CR, LF (0D, 0A)
lfcr	LF, CR (0A, 0D)
nel	NEL (85) (Unicode newline)
lsep	LSEP (2028) (Unicode line separator)
psep	PSEP (2029) (Unicode paragraph separator)
null	NUL (0)

Encoding Values

Most Stylus XML Converters allow you to specify the type of encoding to perform during the conversion. The following table describes the types of encoding you can specify. These values are derived from the character sets available from the currently-running JVM, with a few additions. There are several more values that are available when theStylus XML Converters library is used.

Table 5-2. Encoding Values

Value	Description
raw	The byte values 0x00 to 0xFF are mapped to char values \u0000 to \u00FF. In addition, char values have the high-order 8 bits stripped, producing byte values.
ebcdic	Use this value if your local JVM does not support an EBCDIC code page to allow the data to be sent and returned as EBCDIC. It is based on one of the many internationalized EBCDIC standards, but has an additional property that allows each codepoint to be mapped to exactly one codepoint on the 8-bit plane.

Base-64 XML Converter Properties

The following table lists properties for the Base-64 XML Converter, which can be used for Base-64 encoded binary files as documented in RFC 1341.

XML Converter Name in URI

Base-64

Table 5-3. Properties for the Base-64 XML Converter

Name in URI	Property Name	Description
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is utf-8.
newline	Line separator	Specifies the line separator character. See "Line Separator Values" on page 115 for a list of commonly used values. This property is used only when converting a Base-64 binary file to XML, and not vice versa. The default is crlf.

Binary XML Converter Properties

You can convert binary files that have been encoded as a sequence of digits in a base from 2 to 36, and vice versa.

Use the Base-64 XML Converter for base-64 encoded binary files. See “Base-64 XML Converter Properties” on page 117 for more information.

The following table lists the properties for the Binary Base-2 to Base-36 XML Converter.

XML Converter Name in URI

Binary

Table 5-4. Properties for the Binary Base-2 to Base-36 XML Converter

Name in URI	Property Name	Description
base	Base	Specifies the numeric base of the encoded file. The default is 16 (hexadecimal); Base-2 is binary; Base-8 is octal; and Base-10 is decimal.
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is utf-8.
newline	Line separator	Specifies the line separator character. See “Line Separator Values” on page 115 for a list of commonly used values. This property is used when converting a binary encoded file to XML, and vice versa. The default is crlf.

Table 5-4. Properties for the Binary Base-2 to Base-36 XML Converter

Name in URI	Property Name	Description
space	Byte separator	Determines whether byte values are contiguous (no value) or separated with the value specified for this property. For example, if you set `space=,` the value 000FFF would be output as `00,0F,FF`.
wrap	Wrap lines	Determines whether to wrap lines. Valid values: n yes – lines are wrapped. n no – outputs all values on a single line. The default is yes.

Comma-Separated Values (CSV) XML Converter Properties

You can use the CSV XML Converter to convert comma-separated values files to XML and vice versa.

The following table lists the properties for the CSV XML Converter.

XML Converter Name in URI

CSV

Table 5-5. Properties for the CSV XML Converter

Name in URI	Property Name	Description
collapse	Collapse consecutive separators	Determines whether to collapse consecutive separators, separators that do not contain any data. Valid values: n yes – consecutive separators are collapsed. n no – consecutive separators are not collapsed. The default is no.
double	Doubling embedded quote escapes it	Determines whether doubling an embedded quotation mark has the effect of escaping the quoted string. Valid values: n yes – doubling an embedded quotation mark escapes the quoted string. n no – doubling an embedded quotation mark does not escape the quoted string. The default is no.

Table 5-5. Properties for the CSV XML Converter

Name in URI	Property Name	Description
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is cp1252.
escape	Escape character	Specifies the escape character to be used to escape quotes and separators so that they can be embedded in values. The back slash (\) is the default.
first	First row contains field names	Generated field names depend on the values in the first and number fields. If first=yes and number=no, field names are read from the first row. Any field names after that are named `column.`*`nnn`*, where *`nnn`* is the column number, starting from 1 and including explicitly named columns in the count. If number=yes, extra columns (those after the first) are named column.
multiline=	Multiline	Determines whether a line separator in a quoted string is considered part of the content of a field. Valid values: n yes – a line separator in a quoted string is considered part of the content of that field. n no – a line separator in a quoted string is not considered part of the content of a field and terminates the row, even if it is encountered in the middle of a quoted string. The default is no.
newline	Line separator	Specifies the line separator character. See “Line Separator Values” on page 115 for a list of commonly used values.

Table 5-5. Properties for the CSV XML Converter

Name in URI	Property Name	Description
number	Number rows and columns	Determines whether rows and columns contain a numbering attribute. Valid values: n yes – each row contains an attribute named row that contains the row number from the source document, starting from 1. Also, each column, even those explicitly named, have a column attribute numbering the column from 1. n no – empty columns are omitted from the output, but the numbering of subsequent columns reflects if one or more columns were skipped. The default is no.
quotes	Quote character	Specifies a list of characters that the converter should interpret as quotation marks. The default is double quotes (") and single quotes (’).
root	Root element name	Specifies the root element name. The default is table.
row	Row element name	Specifies the row element name. The default is row.
sep	Separator	Specifies the separator value to appear between each value. This can be TAB, a single character (a comma (,) is the default), or the %*XX*-escaped value of the separator character (%2c, for example).

dBase XML Converter Properties

Properties are the same for all dBase XML Converters: dBase II, dBase III, dBase III+, dBase IV, and dBase V.

The following table lists the properties for the dBase XML Converters.

XML Converter Names in URI

- dBase_II
- dBase_III
- dBase_III_plus
- dBase_IV
- dBase_V

Table 5-6. Properties for the dBase XML Converters

Name in URI	Property Name	Description
deleted	Include deleted records	Determines whether records marked with a "deleted" attribute are included in the output to XML and preserved in the conversion from XML. Valid values: n yes – records marked with a deleted attribute are included in the output to XML. NOTE: Stylus Studio looks for this on input. n no – records marked with a deleted attribute are not included in the output to XML. The default is yes.
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is utf-8.

Table 5-6. Properties for the dBase XML Converters

Name in URI	Property Name	Description
newline	Line separator	Specifies the line separator. See "Line Separator Values" on page 115 for a list of commonly used values. This property is used only to convert a dBase file to XML, not vice versa.

The following table lists the data types supported by the dBase XML Converters for each dBase version.

Table 5-7. Data Type Support for the dBase XML Converters

Data Type	Symbol	dBase II	dBase III	dBase III+	dBase IV	dBase V
binary	B					X
character	C	X	X	X	X	X
date	D		X	X	X	X
float	F				X	X
general	G					X
logical	L	X	X	X	X	X
memo	M		X	X	X	X
numeric	N	X	X	X	X	X

DIF XML Converter Properties

You can use the Data Interchange Format (DIF) XML Converter to convert DIF files to XML and vice versa.

The following table lists the properties for the DIF XML Converter.

XML Converter Name in URI

DIF

Table 5-8. Properties for the DIF XML Converter

Name in URI	Property Name	Description
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is cp850.
newline	Line separator	Specifies the line separator character. See “Line Separator Values” on page 115 for a list of commonly used values. This property is used when converting a DIF file to XML and vice versa. The default is crlf.

EDI XML Converter Properties

You can use the Electronic Data Exchange (EDI) XML Converter to convert EDI files to XML and vice versa.

Properties are the same for most supported EDI dialects; however, some properties are dialect-specific (cexpand and hexpand, for example).

TIP: Stylus XML Converters supports the Standard Exchange Format (SEF) standard, which allows you to define extensions to an EDI standard. See “Handling Proprietary EDI Formats” on page 25 for more information.

The following table lists properties for the EDI XML Converter.

XML Converter Name in URI

EDI

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
auto	Auto-fixup values where possible	Automatically fills in values of segments and elements where possible. Valid values: n never — never autofill values of segments and elements. n toXML — autofill values of segments and elements only when converting from an EDI format to XML. n fromXML — autofill values of segments and elements only when converting from XML to an EDI format. n both — autofill values of segments and elements when converting from an EDI format to XML or vice versa. See "Autofilling Segments and Elements" on page 173 for a list of which segments and elements are autofilled See also "noautofill" on page 147.
bom	Inserting a BOM while writing EDI to the Writer class	Inserts a BOM while writing EDI to the Writer class in Java with UTF-8 or UTF-16 encoding. Valid values: n yes — inserts a BOM while writing. n no — begins writing without inserting a BOM. The default is no.
cent	Window for century cut-off	If the date is given in the file with a 2-digit year and the output requires a 4-digit year, this value is the cutoff so that the proper century can be selected.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
cexpand	Fully expand HL7 CE/CF/CNE/CWE element	HL7 includes specialized composite elements that contain coded values, the text version of the code, and the lookup table for CE, CF, CNE, and CWE elements. Valid values: n yes — the converter attempts to expand all fields in the composite element. n no — the converter does not expand fields in the composite element. The CNE and CWE elements also allow pulling information from tables across versions of the standard. For example, an HL7 2.4 element could look up information from an HL7 2.5 table. These elements also have an alternate set of fields so that codes can be included in the native (HL7, for example) and foreign code list. The default is no.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
chr	Character repertoire override	Allows the converter to override and alter character encodings for EDIFACT-based documents (such as EANCOM and IATA PADIS). You can use one or more of the following values, concatenated with a "+" symbol (`chr=REPLACE+FINNISH`, for example): n DEFAULT – the encoding specified in the file is used. This value cannot be used with any others. n EANCOM – support for these extra EANCOM characters to UNOA and UNOB are added: #, @, [,], {, }, \, |, ‘, and ^. n SYMBOL – forces all characters, including special characters such as element and segment separators, that might otherwise be permitted to be validated against the encoding. n REPLACE – replaces invalid characters with the character specified by the invalid property. An underscore ("_") is used if the invalid property is not specified. If REPLACE is not specified, the converter throws an error. n FINNISH – changes the meaning of certain characters in the Finnish character set for UNOA and UNOB (and adds UNOY and UNOZ as synonyms for UNOA and UNOB respectively). See “FINNISH Character Set Overrides” on page 159 for more information. See “Explicit Character Overrides” on page 160 for more information.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
clean	Remove linefeeds and nulls	Determines whether to remove linefeeds and nulls from EDI that is converted to XML and vice versa. Valid values: n both – removes line feeds and nulls when converting from XML and to XML. n fromXML – removes line feeds and nulls when converting from XML. n toXML – removes line feeds and nulls when converting to XML. n never – never removes line feeds or nulls. The default is both. See also “rtrim” on page 150.
component	Component value separator	Specifies the character that is used to separate component elements from each other within a composite element. This property affects EDI-to-XML conversion and vice versa. See “Using Special Characters for Separators” on page 161 for information about how to specify values for this property.
compress	Compression method for EDI output	For ACORD AL3 only. Determines whether the run-length encoding (RLE) compression method is used for EDI output. Valid values: n none – no compression is used. n rle – RLE compression is used. The default is none.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
continued	Line continuation character	Specifies the character that is appended to the segment terminator when each segment in an EDI message is split onto a new line. This character indicates to the server that the end of the interchange has not been reached. The character is appended to all segments in an interchange, except the last one. This property affects EDI-to-XML conversion and vice versa. This property accepts the same values as "element" on page 135 and "segment" on page 151. NOTE: This property is supported for all dialects except Cargo-IMP.
count	Enforce segment maximum counts	Determines whether the converter enforces segment counts as they are defined in the EDI repository. Valid values: n yes – repository counts are enforced. n no – repository counts are not enforced. n multi – if the repository allows only one instance, it is enforced; otherwise, it treats the count as unlimited. See also "auto" on page 127.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
data	Generate EDI content	Controls whether sample data is generated with XML and EDI sample output files. Valid values: n minimal – the minimum amount of data required to validate the sample file is generated. n none – no sample data is generated. Headers and trailers are generated. n random – random strings are generated; codelist values are selected randomly to fill the data. The default is none. See also “emit” on page 135.
decimal	Decimal character	Specifies the symbol used for the decimal character in the converted file (usually a period or comma). This property affects EDI-to-XML conversion and vice versa. See “Using Special Characters for Separators” on page 161 for information about how to specify values for this property.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
decname	Use DEC mode for all names in these segments	For ACORD AL3 only. AL3 fields can be broken up in two different ways: n DEC names are used to break the field into two smaller fields. n The first character of the field name is used to denote the field type. Use this property to specify the segments in which the field name should be treated as a DEC name. A valid value is a list of one or more segment names. Multiple segment names must be separated by a comma. The default is 5SNG.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
decode	Where to place decoded data values	Specifies where to place code list value descriptions when converting EDI to XML. Valid values are: n no – code list table values are not output as XML: <ISA15><!--I14: Interchange Usage Indicator-->P</ISA15> n comment – adds the description as a comment. For example, <!--Production Data--> in the following code: <ISA15><!--I14: Interchange Usage Indicator-->P<!--Production Data--></ISA15> n attribute – adds the description as an attribute: <ISA15 decode="Production Data"><!--I14: Interchange Usage Indicator-->P</ISA15> n text – adds the code as an attribute, and the description as an element value: <ISA15 value="P">Production Data</ISA15> NOTE: The value property must also be set to attribute to generate this output. Turn off this and *Comment element types* (field) to disable all comment generation. See also “value” on page 157.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
delimiter	Delimiters to be used for quoted strings in flatfiles	Specifies the delimiter to use for quoted strings in flat files. The default is Q, which means you can use either single quotes (’) or double quotes ("). To specify a specific character, use the Unicode value. For example, a value of 0x22 specifies double quotes and a value of 0x27 specifies single quotes.
dialect	EDI dialect	Overrides the dialect that is detected by the converter of the file you are converting. This property affects EDI-to-XML conversion and vice versa. Valid values: n AHM780 n AL3 n CARGO n EANCOM n EDIFACT n EDIG@S n HIPAA n HL7 n IATA n NCPDP n TELCO n TRADACOMS n X12 Use IATA to specify the PADIS dialect. See “XML Schema Generation” on page 30 for more information about how this property affects XML Schema generation. See “HTML/XHTML Documentation Generation” on page 41 for information about how this property affect HTML/XHTML documentation generation.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
doc	Include xs:documentation	Determines whether to include xs:documentation comments in the XML Schema. Valid values: n yes – xs:documentation comments are included in the XML schema. n no – xs:documentation comments are not included in the XML schema. The default is yes. This property is used only for schema generation. See “XML Schema Generation” on page 30 for more information.
element	Element separator	Specifies the character that is used to separate elements in a segment. This property affects EDI-to-XML conversion and vice versa. See “Using Special Characters for Separators” on page 161 for information about how to specify values for this property.
emit	Level of sample data generated	Controls which optional segments are generated in XML and EDI sample output files. Valid values: n none – only mandatory and elements are created. n segments – all segments are created, but only mandatory elements within those segments are populated with sample data. n elements – all segments and all elements are created. The default is none. See also “data” on page 132.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
empty	How to handle empty HL7 content	Controls how the converter manages empty fields. Empty tokens at the first level are never written to the XML file, regardless of how this property is set. In HL7 versions prior to 2.3, empty fields were treated as present, but without a value; in HL7 version 2.3 and later, empty fields are indicated with a set of double quotes. A missing field (a field for which there is no value in the data stream) does not display quotes. Valid values for this property are: n auto – the HL7 version determines how the converter treats empty fields: n For HL7 2.2 and earlier, the converter behaves as if empty=empty. n For HL7 2.3 and later, the converter behaves as if empty=quotes. n empty – all empty fields, with or without quotes, are treated as present, but empty (use for HL7 2.2 and earlier). n quotes – if the field has quotes, it is treated as empty (null value). For example, ′""′ is recognized as a marker for an empty field; otherwise, it is treated as missing, that is, there is no value in the data stream (use for HL7 2.3 and later). The default is auto.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
empty (cont'd)		Consider the following examples for different conversion scenarios: n HL7 to XML, empty=empty - Empty top-level elements are not output. - Empty lower-level elements are output as as empty XML elements. - Elements containing paired double-quotes are passed through unchanged. n HL7 to XML, empty=quotes - Empty top-level elements are not output. - Empty lower-level elements are not output. - Elements containing paired double-quotes are output as empty XML elements. n XML to HL7, empty = empty - Empty elements are passed through as empty. - Elements containing paired double-quotes are passed through unchanged. n XML to HL7, empty = quotes - Empty elements are passed through as paired double-quotes. - Elements containing paired double-quotes are passed through unchanged.
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default depends on the dialect of EDI and information encoded in the particular EDI file.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
eol	Add linefeeds between segments on write	Allows you to put each segment on its own line when converting XML to EDI. (Extra linefeeds are ignored when converting EDI to XML.) Valid values are: n yes – the value specified in the line separator property (newline=) is used to separate each segment. The normal segment output character is also generated. n no –linefeeds between segments are not added. n integer between 1 and 1024 – specifies the number of columns on which to wrap a line. So, for example, eol=80 wraps the line at 80 columns. If you specify an integer, the last row is not padded out to the value you specify. The default is yes.
field	Comment element types	Creates a comment at the start of each element that includes the element's name and number. For example, `<!--I14: Interchange Usage Indicator-->` in the following code: `<ISA15><!--I14: Interchange Usage Indicator-->P<!--Production Data--></ISA15>` Turn off this and *Comment code list* (decode) to disable all comment generation.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
following	Segment name/segment content separator	In TRADACOMS data streams, the default character used to separate the segment name and segment contents is the equal sign (=). You can use the following= property to override the default character. This property affects EDI-to-XML conversion and vice versa. See "Using Special Characters for Separators" on page 161 for information about how to specify values for this property.
group	Use message groups if provided	Allows you to add grouping elements to XML when converting EDI to XML. Grouping elements can make EDI messages in the converted XML easier to access with XPath for some types of documents. You may want to use this property if your EDI documents use multiple message groups. Valid values: n yes – wraps each message group with an extra <GROUP></GROUP> element. n always – wraps all output associated with an interchange (for example, everything from an ISA segment to its IEA segment, inclusive) in an <INTERCHANGE></INTERCHANGE> element. <GROUP></GROUP> elements are also used. n no – grouping elements are not added to the converted XML. The default is no. This property can also be used when generating XML Schema.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
hexpand	Expand HL7 hex escapes	In HL7 data streams, \X is an escape sequence used to include hexadecimal data in the stream. Valid values n yes – expands the hexadecimal data. If the data is binary, an exception is thrown. n no – does not expand the hexadecimal data. The default is no.
hipaa	Enable HIPAA Auto-detection	Determines whether the converter should determine if an X12 file is a HIPAA file. Valid values: n yes – the converter determines whether the X12 file is a HIPAA file. If the file is a HIPAA file, HIPAA rules are used. If not, X12 rules are used. n no – the converter processes the file as an X12 document, even if it is recognized as a HIPAA document. n loop – same as yes, but this value also creates a nested loop structure for the converted XML and generated XML Schema, which can simplify this output's use in XML mapping tools. The default is no.
ignore	Ignore specific errors	Allows you to specify which errors, if any, to ignore during XML conversion. The syntax for this field is `ignore=n1,n2,n3....` For example, `ignore=3,4,47` ignores errors 3, 4, and 47. This property can be used with the opt property to allow processing to continue even when the data stream is missing mandatory segments and data elements.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
indent	Whether to indent XML output	Controls whether the XML output is indented. Valid values: n yes – XML output is indented. n no – XML output is not indented. blank (unspecified) – XML output is indented unless decode and field are both set to no. The default is blank (unspecified).
inter	Interactive messages	Certain EDI messages have alternate batch and interactive forms, depending on whether they are used between systems that have real-time connections. Valid values: n yes – the interactive form is used, if available. For example, in EDIFACT, the normal envelope of UNB/UNH/UNT/UNZ would be replaced by UIB/UIH/UIT/UIZ. n no – the alternate batch form is used. The default is no. This property is used only for schema generation. See “XML Schema Generation” on page 30 for information on how this property affects XML Schema generation. See “HTML/XHTML Documentation Generation” on page 41 for information about how this property affects HTML/XHTML documentation generation.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
intra	Check intra-segment constraints	Controls whether intra-segment validation is performed for EDIFACT and X12 files. Valid values: n yes – Intra-segment validation is performed. n no – Intra-segment validation is not performed. The default is no.
invalid	Invalid character replacement	Used with the chr =REPLACE property to specify which character is used to replace invalid characters. This property affects EDI-to-XML conversion and vice versa. Valid values: n \u#### – To specify a Unicode value, substituting the #### for the appropriate value. n \d#### – To specify a decimal value, substituting the #### for the appropriate value. The default is an underscore ("_"). See "Using Special Characters for Separators" on page 161 for more information about how to specify values for this property.
iso	Format date and time in the XML output	Formats the date and time in the XML output. Valid values: n yes – date and time are formatted according to the ISO 8601 date and time formats. n no – date and time are not formatted. The default is no.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
ldate	How to handle 'L' HL7 date	Controls how the converter manages the *L* value (traditionally means "local system" in HL7) if it is passed as a date, time, datetime, or timestamp. Valid values: n header – The *L* value is replaced with the value of the MSH-7 element from the header. n current – The *L* value is replaced with the date and/or time that the message processing started. n error – The *L* value is treated as a syntax error. n pass – The *L* value is passed through unchanged.
leading	Ignore leading zeros on numbers	Determines whether the converter ignores leading zeros on numbers. Valid values: n yes – leading zeros on the value in the EDI and the value in the codelist are compared without any leading zeros. For example, a value of "012" matches a value of "12." This applies only to codelist validation, not to handling of numbers. n no – leading zeros are not ignored. The default is no.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
len	Strict validation on value lengths	For most EDI dialects, controls whether an element's content is checked against the upper and lower length limits as defined by the relevant EDI specification. For Cargo-IMP, however, len= enables or disables checking the entire message length against the standard limit of 1600 characters. Because Cargo-IMP requires fixed positions of certain elements, element length checking is always performed. Valid values: n yes – length checking is enabled. n no – length checking is disabled. The default is no.
long	Use long element names	Determines whether to use long or short element and/or segment names in your XML conversions (FTX03-TextReference or FTX03, for example). Valid values are: n elements – long names are used for elements. (In previous versions, long=yes could be used. For naming consistency, long=yes has been deprecated.) n segments – long names are used for segments. n all – long names are used for both elements and segments. n none – abbreviated names are used for both. The default is none.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
loop-prefix	Prefix GROUP_... tags with the enclosing message name	Allows you to prefix the name of a GROUP_*no* tag with the message name it appeared in. For example, `<INVOIC>…<GROUP_1> becomes < INVOIC >…< INVOIC _GROUP_1>`. The default is no.
loopnames	Use named loops	You can name loops (segment groups). Loop names appear in both auto-reply messages, XML output, and HTML/XHTML generated documentation. Valid values: n yes – naming for loops is turned on. n no – naming for loops is turned off. The default is no.
match	Get field count based on input row length	For ACORD AL3 only. Determines how the field count is determined. Valid values: n yes – preserves partial fields at the end of segments by not trimming spaces even from ZZZZZ elements. n no – the repository determines segment size. The default is no.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
message	Message type	The type of EDI message being converted. Valid values vary based on dialect and version. See "XML Schema Generation" on page 30 for information about how this property affects XML Schema generation. See "HTML/XHTML Documentation Generation" on page 41 for information about how this property affects HTML/XHTML documentation generation. NOTE: Normally, you do not need to specify this property, but it is automatically gathered from the incoming stream. If you want to force the incoming message to be processed as another message, you can use this property to specify the alternate message. This does not mean that the conversion will fail if the incoming message does not equal this value; rather, it forces the incoming message to be processed as if the incoming message *were* this value. For an example code that would stop processing if the incoming message was not the message that was expected, see ??????.
newline	Line separator	Used when converting EDI to XML, and XML to EDI when the *Add linefeeds between segments on write* property (eol) is set to yes. See "Line Separator Values" on page 115 for a list of commonly used values. The default is crlf.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
noautofill	Segments and/or elements to not auto-fill	Specifies segments and elements that you do not want the converter to autofill (see "auto" on page 127). This property accepts a comma-separated list of segments or elements. n To specify segments, type the segment name followed by an asterisk (`noautofill=UNB*`, for example). n To specify elements, use the short form of the element name (`noautofill=UIZ01`, for example). See "Autofilling Segments and Elements" on page 173 for a list of which segments and elements are autofilled
nochange	Unchanged fields with how many ?'s	For ACORD AL3 only. The ACORD standard provides two ways of marking fields that you want left at their current value on the receiving system. Valid values: n single – marks only the first character of a field with a question mark (?). n fill – marks all characters in a field with a question mark (?). The default is single.
numpad	Pad numbers with alternate characters	Specifies an alternate character to pad numbers in fixed-width fields if syntactically valid. NOTE: In a flat file conversion using the GENERIC converter, numbers may get padded with the NUL character (0x00). In this case, specify `numpad=NUL`.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
opt	Treat all segments and elements as optional	Determines whether mandatory segments and elements are treated as optional, which can be useful if your provider declines to provide segments and elements that are considered mandatory according to the EDI specification and you know which segments and elements are missing. Valid values: n yes – all mandatory segments and mandatory data elements are treated as optional. n no – a missing mandatory segment or mandatory data element triggers an error. NOTE: You can use opt=no with the ignore property to ignore errors for missing mandatory segments (errors 39 and 9), missing mandatory data elements (error 4), or both. For example, opt=no and ignore=39,9 allows processing to continue even if the data stream is missing mandatory segments. The default is no.
prefix	Namespace prefix	Specifies the namespace prefix to be added, with the Namespace URI, to the root element. The prefix is added to all elements.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
release	Release (escape) symbol	Specifies the release, or escape, character. It turns off special processing of the next character. Suppose your text message uses the same character that also was used to separate elements. The specified character is used to instruct the EDI processor to treat that character as a normal character and not as the end of the text. This property affects EDI-to-XML conversion and vice versa. See "Using Special Characters for Separators" on page 161 for information about how to specify values for this property.
reorder	Sort and nest groups by inferred level	For ACORD AL3 only. Data segments in AL3 files can be in any sequence. This property controls the sequence of the data stream. Valid values: n as-is – the sequence of the data stream is not changed. n nest – the data stream is scanned for segment levels and the stream is sorted based on the implied message structure. n sort – the data stream is sorted based on information in the segment header. The default is as-is.
repeat	Repeat symbol	Specifies the repeat symbol for EDI dialects that use it. This property affects EDI-to-XML conversion and vice versa. See "Using Special Characters for Separators" on page 161 for information about how to specify values for this property.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
rtrim	Trim trailing delimiters	Typically, most trailing delimiters are removed automatically. But in the conversion process, new trailing delimiters are sometimes created. This property controls how the converter manages these trailing delimiters. Valid values: n no – trailing delimiters are not trimmed. n yes – trailing delimiters are trimmed as long as performance is not affected. n always – trailing delimiters are trimmed regardless of the impact on performance. You can also use this property to *add* additional spaces or padding: n pad1 – add spaces at the top-most level only. n pad2 – add spaces to the first two levels only (elements and composite elements that do not contain other composites). n pad3 – adds paces for every level of element. This value can create many empty elements (when converting to XML) and many empty delimiters (when converting to EDI). See also “clean” on page 130.
seg	Strict segment-ordering checking	Determines whether to check segment ordering as defined by the message. Valid values: yes – checks segment ordering. no – message and group definitions are ignored, which can cause data to be grouped incorrectly (<GROUP_*n*> tags are never emitted, for example). The default is yes.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
segment	Segment separator	Specifies the character to use for segment separators. This property affects EDI-to-XML conversion and vice versa. See "Using Special Characters for Separators" on page 161 for information about how to specify values for this property.
setup	Write setup segment if appropriate	When converting EDI to XML for EANCOM, EDIFACT, Edig@s, IATA, and NCPDP SCRIPT, you can generate UNA segments in the XML. This segment is also recognized when converting XML back to EDI as an alternative to using the segment= URI option. Valid values: n both – converts UNA segments in EDIFACT to XML and vice versa. Also generates an XML Schema for UNA when the schema generator is used. n default – as opposed to turning the emitting of a setup segment on or off explicitly, the converter uses the default value for the dialect. For example, for EDIFACT, the default segment is "from XML." For other dialects, the converter may generate the header segment in both directions. n toXML – converts UNA segments in EDIFACT to XML and generates an XML Schema when the schema generator is used. n fromXML – converts UNA elements in XML to UNA segments in EDI. n never – does not convert UNA segments to XML, or vice versa. The default is both.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
sign	Use explicit '+' sign to denote positive numbers	For ACORD AL3 only. When writing AL3, the default for denoting a positive number is the space character. This property explicitly sets the plus sign (+) to denote positive characters. Valid values: n yes – the plus sign (+) is used to denote positive numbers. n no – the space character is used to denote positive numbers. The default is no.
strict	Strict validation mode	Determines whether to perform strict validation checking. Valid values: n yes – checks that all mandatory elements are present and ensures that no composite elements are in places where only simple elements are allowed. It also checks for extra elements at the end of segments that are not part of the specification. n no – does not perform strict validation. The default is no.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
strip	Strip C-style comments	Determines whether content in the incoming EDI stream that is wrapped in C-style `/*` and `*/` comment delimiters is ignored. Valid values: n yes – ignores C-style comment delimiters. n no – does not ignore C-style comment delimiters. The default is no. NOTE: The default setting is no because ignoring the delimiters can result in a conflict with real EDI content. HL7 files used with or generated by certain systems may include this markup, for example.
syntax	Force the syntax level of the EDIFACT-style input	Determines the syntax level of the input. Valid values: n yes – allows EDIFACT-style input to be overridden from the assumed value in the UNB0101/0001 element. Also, it is especially useful when the input stream does not contain a UNB or UIB segment and one must be created. n no – does not allow EDIFACT-style input to be overridden. The default is no.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
ta1	Include TA1 segment	Allows inclusion of the TA1 segment in the 999 reply. Valid values: n no – does not generate the TA1 segment in the reply. n yes – always generates the TA1 segment in the reply. n auto – uses the ISA14 setting from the incoming interchange to determine whether to add the TA1 segment in the reply. The default is no.
tbl	Force error if value not in code list	Determines whether the converter checks the value of an element for its presence in the codelist associated with that element. Valid values: n yes – values are checked for their presence in the codelist. If they are not found in the codelist, an error is generated. n no – values are not checked for their presence in a codelist. The default is no.
terminate	Stop reading the input	Specifies a character that stops the reading of the input during conversion. If the converter encounters the specified character, the input streaming stops at that character. This can be useful to define the end-of-file marker so that no data after this point will be read. This property only applies when reading from EDI and not for parsing XML.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
tertiary	Subcomponent (tertiary) separator	Specifies the character to use for subcomponent separators. This property affects EDI-to-XML conversion and vice versa. See "Using Special Characters for Separators" on page 161 for information about how to specify values for this property.
toobig	Proper tuning of large segments	For some dialects, such as HL7, it is common to have large segments that can contain more than 100 elements. Normally, the error recovery algorithm is applied when large segments are encountered, and the converter assumes that separators are specified incorrectly or that the input is corrupt. This property specifies a limit on the number of elements.
typ	Strict datatype content checking	Ensures that only characters that are appropriate for a given field are included in the value for that field. For example, this property ensures that dates are valid and numbers are well-formed.
unbounded	Upper limit on maxOccurs, or unbounded	Determines whether an upper threshold is enforced on maxOccurs. For example, if `unbounded=50`, all occurrences of maxOccurs in an XML schema that have a value of 50 or higher are changed to unbounded. Valid values: n A positive integer greater than 1 – sets the upper threshold on maxOccurs. n unbounded – does not set an upper threshold on maxOccurs. The default is unbounded.
uri	Namespace URI	Specifies a namespace URI to be added, with the Namespace prefix, to the root element. If the prefix is set, but the URI is not, the prefix is ignored.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
user	Extension map file	Specifies the URL of the SEF file containing custom message type definitions. This property can also be used when generating an XML Schema.
val	Enable validation	Determines whether validation is enabled. Valid values: n yes – validation is enabled. The version, release, messages and segments of the EDI file (input or output) is compared to the relevant EDI repository. If the EDI file contains a value that is not in the EDI repository, an error is thrown. n no – validation is disabled. Processing continues even if the EDI file contains a version, release, message, or segment that is not in the repository. When processing an unknown version, release, message, or segment, some checks cannot be performed because the structure of the required data is unknown. For example, if a file is of a known version but contains an unknown segment, data type checking for that segment is not performed, but checks on the remainder of the file are performed as usual. Similarly, if a message does not exist in the EDI repository, the file is still processed, but segment order checking is not performed. The default is yes.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
value	Where to place coded data values	Allows you to specify where to place coded data values in XML output. Valid values: n text – outputs the coded data value (here, 00) in the text node: <BGN01><!--353: Transaction Set Purpose Code-->00</BGN01> n attribute – outputs the coded data value as an attribute: <BGN01 value="00"><!--353: Transaction Set Purpose Code--></BGN01> See also “decode” on page 133.
version	Dialect version	Overrides the version of the input file that is detected by the converter. This property affects EDI-to-XML conversion and vice versa. Valid values vary based on the specified dialect (see“dialect” on page 134). See “XML Schema Generation” on page 30 for more information about how this property affects XML Schema generation. See “HTML/XHTML Documentation Generation” on page 41 for more information about how this property affects HTML/XHTML documentation generation.

Table 5-9. Properties for the EDI XML Converter

Name in URI	Property Name	Description
xr	Type of transaction message to be generated	Specifies the type of transaction message to be generated. For example, if set to a value of 999, a 999 response message is generated instead of generating the 997 response message. Valid values: n 997 – a 997 response message will be generated for transactions. n 999 – a 999 response message will be generated for transactions. NOTE: If you specify `xr=999`, make sure that the version of X12 being used supports the 999 response message, which was introduced in version 005010. This property is HIPAA-aware; therefore, for version 0050x0, it writes the 005010X231 version of 999, and for 0060x0, it writes the 006010X290 version. The default is 997.
zz	Include ZZMOV and/or ZZDEL in output	For ACORD AL3 only. Affects whether moved (ZZMOV) or deleted (ZZDEL) fields in a file are emitted to XML. Valid values: n none – no ZZ* elements are emitted. n delete – only ZZDEL elements are emitted. n move – only ZZMOV elements are emitted. n all – all ZZ* elements are emitted. The default is none.

FINNISH Character Set Overrides

You can use the Character repertoire override property (see "chr" on page 129) to change the meaning of certain characters for the Finnish character set for UNOA (UNOY) and UNOB (UNOZ). The following table shows which characters are changed based on the character set in use.

Table 5-10. Character Encoding Overrides

Character From	Character To	Applicable Character Sets	Unicode Name
[	Ä	UNOA/UNOY, UNOB/UNOZ	Latin capital letter A with diaeresis
\	Ö	UNOA/UNOY, UNOB/UNOZ	Latin capital letter O with diaeresis
]	Å	UNOA/UNOY, UNOB/UNOZ	Latin capital letter A with ring above
^	Ü	UNOA/UNOY, UNOB/UNOZ	Latin capital letter U with diaeresis
{	ä	UNOB/UNOZ	Latin small letter a with diaeresis
|	ö	UNOB/UNOZ	Latin small letter o with diaeresis
}	å	UNOB/UNOZ	Latin small letter a with ring above
~	ü	UNOB/UNOZ	Latin small letter u with diaeresis

Explicit Character Overrides

In addition to the modifiers you can specify using the Character repertoire override property (see "chr" on page 129), you can instruct the Stylus XML Converters to take character encodings from the URI, instead of from the EDIFACT-style UNB or UIB 001 element. If you choose to do this, you can use the encodings described in the following table:

Table 5-11. Character Encoding Overrides

Property Name	*Description*
UNOA or IATA	UN/ECE level A (upper case only)
UNOB or IATA	UN/ECE level B (same as UNOA but including lower case)
UNOC or IATC	UN/ECE level C (ISO-8859-1 or Latin-1/Western European)
UNOD or IATD	UN/ECE level D (ISO-8859-2 or Latin-2/Central European)
UNOE or IATE	UN/ECE level E (ISO-8859-5 or Latin/Cyrillic)
UNOF or IATF	UN/ECE level F (ISO-8859-7 or Latin/Greek)
UNOG or IATG	UN/ECE level G (ISO-8859-3 or Latin-3/South European)
UNOH or IATH	UN/ECE level H (ISO-8859-4 or Latin-4/North European)
UNOI or IATI	UN/ECE level I (ISO-8859-6 or Latin/Arabic)
UNOJ or IATJ	UN/ECE level J (ISO-8859-8 or Latin/Hebrew)
UNOK or IATK	UN/ECE level K (ISO-8859-9 or Latin-5/Turkish)
UNOQ or IATQ	UN/ECE level Q (ISO-8859-15 or Latin-9/)
UNOW or IATW	UN/ECE level W (ISO 10646-1 octet with code extension technique to support UTF-8)
UNOX	Unsupported. An external library must be used to handle the encoding prior to supplying the data stream to the Stylus XML Converters.
UNOY or IATY	UN/ECE level Y (ISO 10646-1 octet without code extension technique.); also Finnish UNOA
UNOZ or IATZ	Finnish UNOB

These can also be combined with other chr= options. For example, an EDI file might specify an UNOA encoding, but with lower-case

text, because the sending system sent inconsistent data. Using `chr=UNOB+REPLACE`, the data could be consumed, and any non-UNOB characters would turn into '_' characters, allowing processing to continue.

The Stylus XML Converters do not perform character checking for UNOX; rather it depends on the native platform converter or the application to ensure that the characters are valid. This is because there are too many implementation-specific details, subsets, and proprietary and local extensions, and it is not possible to account for them all.

Using Special Characters for Separators

Most special characters or symbols cannot be entered directly into a URL. For example, you cannot specify that the colon (:) is the element separator character by entering `converter:EDI:element=::auto=both`. Instead, you must escape special characters using the appropriate decimal or hexadecimal value. To specify a colon as an element separator character, you would use `converter:EDI:element=\u3A:auto=both`.

NOTE: The special characters that are supported vary depending on the specified EDI dialect.

See Table 5-12, “Common Separator Characters,” on page 163 for a complete list of separator characters and their decimal and hexadecimal values.

Which Properties Specify Separators?

The following properties can be used to specify separators for the EDI XML Converter:

Restrictions for Separator Characters

The following restrictions apply for using separator characters.

- The values you set for separator properties apply only when converting XML to EDI.
- You cannot use letters, numbers, or spaces for separator characters.
- You must use unique values for each separator property.

Commonly Used Separator Characters

Commonly used separator characters and their escape values (in decimal and hexadecimal) are shown in the following table.

Table 5-12. Common Separator Characters

Character	Decimal	Hexadecimal
~	\d126	\u007E
!	\d33	\u0021
@	\d64	\u0040
#	\d35	\u0023
$	\d36	\u0024
%	\d37	\u0025
^	\d94	\u005E
&	\d38	\u0026
*	\d42	\u002A
(	\d40	\u0028
)	\d41	\u0029
_	\d95	\u005F
+	\d43	\u002B
`	\d96	\u0060
-	\d45	\u002D
=	\d61	\u003D
[	\d91	\u005B
]	\d93	\u005D
}	\d123	\u007B
{	\d125	\u007D
\	\d92	\u005C
|	\d124	\u007C
’	\d39	\u0027
;	\d59	\u003B
"	\d34	\u0022

Table 5-12. Common Separator Characters

Character	Decimal	Hexadecimal
:	\d58	\u003A
/	\d47	\u002F
.	\d46	\u002E
,	\d44	\u002C
?	\d63	\u003F
>	\d62	\u003E
<	\d60	\u003C

Control Characters

You can also use the non-printable control characters shown in the following table as separators.

Table 5-13. Control Characters

Character	Decimal	Hexadecimal	Other
NUL	\d0	\u0000	
SOH	\d1	\u0001	
STX	\d2	\u0002	
ETX	\d3	\u0003	
EOT	\d4	\u0004	
ENQ	\d5	\u0005	
ACK	\d6	\u0006	
BEL	\d7	\u0007	
BELL	\d7	\u7000	
BS	\d8	\u0008	
HT	\d9	\u0009	\t
TAB	\d9	\u0009	\t

Table 5-13. Control Characters

Character	Decimal	Hexadecimal	Other
LF	\d10	\u000A	\n
VT	\d11	\u000B	
FF	\d12	\u000C	\f
CR	\d13	\u000D	\r
SO	\d14	\u000E	
SI	\d15	\u000F	
DLE	\d16	\u0010	
DC1 (XON)	\d17	\u0011	
DC2	\d18	\u0012	
DC3 (XOFF)	\d19	\u0013	
DC4	\d\0	\u0014	
NAK	\d21	\u0015	
SYN	\d22	\u0016	
ETB	\d23	\u0017	
CAN	\d24	\u0017	
EM	\d25	\u0019	
SUB	\d26	\u001a	
ESC	\d27	\u001b	
FS	\d28	\u001c	
GS	\d29	\u001d	
RS	\d30	\u001e	
US	\d31	\u001f	
DEL	\d127	\u007F	
BPH	\d130	\u0082	
NBH	\d131	\u0083	
IND	\d132	\u0084	
NEL	\d133	\u0085	
SSA	\d134	\u0086	
ESA	\d135	\u0087	

Table 5-13. Control Characters

Character	Decimal	Hexadecimal	Other
HTS	\d136	\u0088	
HTJ	\d137	\u0089	
VTS	\d138	\u008A	
PLD	\d139	\u008B	
PLU	\d140	\u008C	
RI	\d141	\u008D	
SS2	\d142	\u008E	
SS3	\d143	\u008F	
DCS	\d144	\u0090	
PU1	\d145	\u0091	
PU2	\d146	\u0092	
STS	\d147	\u0093	
CCH	\d148	\u0094	
MW	\d149	\u0095	
SPA	\d150	\u0096	
EPA	\d151	\u0097	
SOS	\d152	\u0098	
SCI	\d154	\u009A	
CSI	\d155	\u009B	
ST	\d156	\u009C	
OSC	\d157	\u009D	
PM	\d158	\u009E	
APC	\d159	\u009F	
NBS (NBSP)	\d160	\u00A0	
SHY	\d173	\u00AD	

EDI Processing Instructions

You can specify EDI processing instruction (PI) values in the EDI XML Converter URI as described in the following table.

Table 5-14. Properties for EDI Processing Instructions

Processing Instruction	Name in URI	Description	Default
edi_component	component	Component value separator.	:
edi_continued	continued	Line continuation character	No default
edi_decimal	decimal	Decimal character.	,
edi_delimiter	delimiter	Delimiter for quoted strings	Q
edi_element	element	Element separator.	+
edi_following	following	Segment name/segment content	=
edi_invalid	invalid	Invalid character replacement	_
edi_numpad	numpad	Pad numbers with alternate characters	No default
edi_release	release	Release (escape) separator.	?
edi_repeat	repeat	Repeat symbol separator.	~
edi_segment	segment	Segment separator.	’
edi_terminate	terminate	Stop processing character	No default
edi_tertiary	tertiary	Subcomponent (tertiary) separator.	&

Leave these values blank to assume the default values. Stylus XML Converters generates an error if a PI and URI switch have conflicting values, or if either value conflicts with one of the values encoded in a segment for these values.

The syntax of an EDI processing instruction is <?, followed by processing instruction name (edi_segment, for example), followed by a space, and then the new special character.

Example

Suppose an X12 document had to be written so that the segment terminator was a carriage return, the element separator was an asterisk, and the component separator was the greater-than symbol. The start of the file might look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<?edi_segment \r?>
<?edi_element *?>
<X12>
   <ISA>
      <ISA01><!--I01: Authorization Information Qualifier-->00</ISA01>
      <ISA02><!--I02: Authorization Information--></ISA02>
      <ISA03><!--I03: Security Information Qualifier-->00</ISA03>
      <ISA04><!--I04: Security Information--></ISA04>
      <ISA05><!--I05: Interchange ID Qualifier-->01</ISA05>
      <ISA06><!--I06: Interchange Sender ID-->1515151515</ISA06>
      <ISA07><!--I05: Interchange ID Qualifier-->01</ISA07>
      <ISA08><!--I07: Interchange Receiver ID-->5151515151</ISA08>
      <ISA09><!--I08: Interchange Date-->041201<!--2004-12-01--></ISA09>
      <ISA10><!--I09: Interchange Time-->1217</ISA10>
      <ISA11><!--I65: Repetition Separator-->U</ISA11>
      <ISA12><!--I11: Interchange Control Version-->00403</ISA12>
      <ISA13><!--I12: Interchange Control Number-->000032123</ISA13>
      <ISA14><!--I13: Acknowledgment Requested-->0</ISA14>
      <ISA15><!--I14: Usage Indicator-->P</ISA15>
      <ISA16><!--I15: Component Element Separator-->></ISA16>
   </ISA>
...
```

Stopping a Conversion If the Input Does Not Match What is Expected

You may want to terminate the input if the input contains an incorrect message. If the auto-detection feature incorrectly identifies the message type from the incoming data, you can use the message= URI property to force the incoming message to act in a different way.

To quickly terminate the input, a "detector" program can be used. The following program reads a sufficient amount of data from an EDI file. It determines if the file matches the given specifications, and if the file does not, the conversion is quickly terminated.

Example

The program uses the Stylus XML Converters library and the listener interface. It works because the Stylus XML Converters are designed to stream data. When the converters see input, they immediately write the output. The program reads data and looks for the dialect, version, and message properties, and terminates the conversion as soon as these properties are seen. The system then discards the output obtained from the conversion process.

```
import java.io.IOException;
import java.io.Writer;

import javax.xml.transform.Result;
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;

import com.ddtek.xmlconverter.Converter;
import com.ddtek.xmlconverter.ConverterFactory;
import com.ddtek.xmlconverter.adapter.edi.EDIConverterListener;
import com.ddtek.xmlconverter.adapter.edi.EDISegmentDetails;
import com.ddtek.xmlconverter.exception.ConverterException;

/**
```

```
 * This program will accept one or more EDI files as arguments.
 * First, it processes them until it sees the first StartMessage event,
 * writing the results to null output.
 * Next, it throws a benign exception to stop processing and
 * checks the dialect, version, and message against what is expected.
 * If they match, it restarts processing and transforms the message into an
 * XML file with the same base name as the input file.
 * If they do not match, it stops processing as if an error were encountered.
 */
public class Stopper {

// public static final String DIALECT = "X12";
// public static final String VERSION = "004030";
// public static final String MESSAGE = "831";
   public static final String DIALECT = "HL7";
   public static final String VERSION = "2.1";
   public static final String MESSAGE = "ORU";

   public static void main(String[] args) {
      for (String arg : args)
           new Stopper(arg);
   }

   public Stopper(String arg) {
      ConverterFactory factory = new ConverterFactory();
      Converter toXml;
         try {
            toXml = factory.newConvertToXML("converter:EDI:tbl=no:opt=yes");
         } catch (ConverterException ce) {
            ce.printStackTrace();
            return;
         }
         EDIConverterListener listener = new MessageListener();

         String out = null;
            // yields  "xml" if the input has no '.'
         try {
            out = arg.substring(0, arg.indexOf('.') 1) "xml";

            Source converterSource;
            Result converterResult;
```

```
            // the first pass is to check for conformity
            try {
               toXml.getConfiguration().setConverterListener(listener);
               converterSource = new StreamSource(arg);
               converterResult = new StreamResult(new NullWriter());
               toXml.convert(converterSource, converterResult);
               throw new StopException(null);
            // make sure we handle case where EDI contains no message
            } catch (StopException exception) {
               if (   !isSame(DIALECT, exception.dialect)
                      || !isSame(VERSION, exception.version)
                      || !isSame(MESSAGE, exception.message)) {
                      System.out.println("[Stopper] error! " arg " is not a
                      " DIALECT " " VERSION " " MESSAGE);
                      return;
               }
            }

            // yes, it's okay to re-use the converter
            toXml.getConfiguration().setConverterListener(null);
            converterSource = new StreamSource(arg);
            converterResult = new StreamResult(out);
            toXml.convert(converterSource, converterResult);

            System.out.println("[Stopper] completed: " arg " -> " out);
         } catch (ConverterException exception) {
            System.out.println("[Stopper] failed: " arg " -> " out);
            System.out.println("[Stopper] failed with exception:
            " exception);
         }

   }

   private boolean isSame(String one, String two) {
      if (one == two)
         return true; // catches both same, or both null
         if (one == null || two == null)
            return false; // catches either null
         return one.replaceAll("[-. ]", "").
         equals(two.replaceAll("[-. ]", ""));
```

```
   }

   public static class StopException extends ConverterException {
      public String dialect;
      public String version;
      public String message;
      public StopException(EDISegmentDetails details) {
         super("");
         dialect = details == null ? null : details.getDialect();
         version = details == null ? null : details.getMessageVersion();
         message = details == null ? null : details.getTransactionSet();
      }
         private static final long serialVersionUID = -98684742021305872L;
   }

   public static class NullWriter extends Writer {
      public void write(char[] cbuf, int off, int len) throws IOException { }
         public void flush() throws IOException { }
         public void close() throws IOException { }
    }

   public static class MessageListener implements EDIConverterListener {
      public void error(ConverterException exception) throws
ConverterException {
   throw exception; // take this out to make recoverable errors recover
      }
   public void fatalError(ConverterException exception) throws
ConverterException {
      // no matter what we do here, an exception is thrown by the caller
      }
      public void warning(ConverterException exception) throws
ConverterException {
      System.out.println("[Stopper] warning: " exception.toString());
      }

      public void startInterchange(EDISegmentDetails details) throws
Exception { }
      public void startGroup(EDISegmentDetails details) throws Exception { }
      public void startMessage(EDISegmentDetails details) throws Exception {
         throw new StopException(details); // we've seen enough, now exit
      }
```

```
      public void processSegment(EDISegmentDetails details) throws Exception
{ }
      public void endMessage(EDISegmentDetails details) throws Exception { }
      public void endGroup(EDISegmentDetails details) throws Exception { }
      public void endInterchange(EDISegmentDetails details) throws Exception
{ }
      public int invalidCharacter(char chr, String coding) {
      return -1; // let it throw!
      }
         public String unknownCodelistValue(String segment, int pos, int sub,
int tri, int rep, String element, String item) {
         return null; // let it throw!
         }
   }
}
```

Autofilling Segments and Elements

By default, the values of segments and elements are autofilled during the conversion from XML to an EDI format or vice versa. You can change the default behavior by setting the following XML EDI Converter URI properties:

- auto property determines whether autofilling is enabled or disabled. You can enable autofilling for the following types of conversions:

  - From the EDI format to XML (`auto=toXML`)
  - From the XML format to the EDI format (`auto=fromXML`)
  - From either the EDI format or the XML format (`auto=both`)

  To disable autofilling, use `auto=never`. When autofilling is enabled, the values of elements that are autofilled depend on the specified dialect as described in Table 5-15.

- noautofill property excludes specific segments or elements from being autofilled when autofilling is enabled. For example, to autofill all segments and elements except the UNB and UNZ segments, you could use the following Converter URI fragment:

```
auto=both:noautofill=UNB*,UNZ*
```

Table 5-15. Autofilled Elements For Each Dialect

dialect Property Value	Elements
AHM780	None.
AL3	If the group (segment) header is not supplied, sequence numbers are autofilled.
CARGO	None.
EANCOM	UIB0801, UIB0802, UIT01, UIT02, UIZ0101, UIZ0102, UIZ0103, UIZ0104, UIZ02, UIZ03, UNB0401, UNB0402, UNG0401, UNG0402, UNT01, UNT02, UNZ01, and UNZ02.
EDIFACT	UIB0801, UIB0802, UIT01, UIT02, UIZ0101, UIZ0102, UIZ0103, UIZ0104, UIZ02, UIZ03, UNB0401, UNB0402, UNG0401, UNG0402, UNT01, UNT02, UNZ01, and UNZ02.
EDIG@S	UIB0801, UIB0802, UIT01, UIT02, UIZ0101, UIZ0102, UIZ0103, UIZ0104, UIZ02, UIZ03, UNB0401, UNB0402, UNG0401, UNG0402, UNT01, UNT02, UNZ01, and UNZ02.
HIPAA	CTT01, CTT02, G8501 (calculated CRC value), GE01, GE02, GS04, GS05, IEA01, IEA02, ISA01 (defaults to 00 if missing), ISA03 (defaults to 00 if missing), ISA05 (defaults to ZZ if missing), ISA07 (defaults to ZZ if missing), ISA09 (defaults to YYMMDD if missing), ISA10 (defaults to HHMM if missing), ISA11, ISA12, ISA16 (component element separator, if specified in URI), SE01, and SE02.
HL7	All CE/CF/CNE/CWE-type elements (controlled by cexpand= URI property), MSH.1, and MSH.2
IATA	UIB0801, UIB0802, UIT01, UIT02, UIZ0101, UIZ0102, UIZ0103, UIZ0104, UIZ02, UIZ03, UNB0401, UNB0402, UNG0401, UNG0402, UNT01, UNT02, UNZ01, and UNZ02.

Table 5-15. Autofilled Elements For Each Dialect *(cont.)*

dialect Property Value	Elements
NCPDP	UIB0801, UIB0802, UIT01, UIT02, UIZ0101, UIZ0102, UIZ0103, UIZ0104, UIZ02, UIZ03, UNB0401, UNB0402, UNG0401, UNG0402, UNT01, UNT02, UNZ01, and UNZ02.
TELCO	Values for the COUNT and COUNTING fields are calculated based on the number of elements.
TRADACOMS	ATR01, DNA01, END01, EOB01, MHD01, MTR01, PAT01, RSG01, RSG02, STX0401, and STX0402. Also, if any of the following elements appear, they are autofilled: APSE, APSI, ASDA, ASDT, CONA1, CTOT1, EVLA, EVLT, EXLV, FASE, FASI, FASU1, FBAB1, FPSE, FPSI, FPSU1, FTAK, FTAR, FTCO, FTDE, FTNA, FTNC, FTND, FTNE, FTNI, FTNP, FTNS, FTOP1, FTOR, FTPC, FTSR, FTUP, FVAT, ISTO, LACK, LCON, LDEL, LOCD, LORD, LUPL, LVLA, LVLT, NOLR, NOPP, NOPR, NOTX, PTOT1, SDCD, SEDT, SEQA, SEQB, SEQC, SEQD, SLAJ, SLSN, SRAP, SRDT, SRLC, SRVT, STLD, STLN, STPT, TBTL1, TOTL, TOTV, TPSE, TPSI, TVAT, TVLC, TVLD, TVLP, UCSI1, UPSI1, USDI1, UTVA1, UVAT1, UVLA1, UVLT1, UVTT1, VATA, VPSE, VPSI, VSDE, VSDI, VTVC1, and VVAT.
X12	CTT01, CTT02, G8501 (calculated CRC value), GE01, GE02, GS04, GS05, IEA01, IEA02, ISA01 (defaults to 00 if missing), ISA03 (defaults to 00 if missing), ISA05 (defaults to ZZ if missing), ISA07 (defaults to ZZ if missing), ISA09 (defaults to current date in YYMMDD if missing), ISA10 (defaults to HHMM if missing), ISA11, ISA12, ISA16, SE01, and SE02.

HTML Converter Properties

You can use the HTML XML Converter to convert HTML to XHTML. The following table lists the properties for the HTML XML Converter.

XML Converter Name in URI

HTML

Table 5-16. Properties for the HTML XML Converter

Name in URI	Property Name	Description
encoding	Encoding	The encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is utf-8.
errors	Abort on errors found	Determines whether the converter will fail when it encounters problems with the HTML-to-XHTML mapping. Valid values: n yes – the converter fails when it encounters problems with the mapping. n no – even if the converter encounters problems with the mapping, the converter continues with the conversion, making a best guess. The default is yes.

Table 5-16. Properties for the HTML XML Converter

Name in URI	Property Name	Description
newline	Line separator	Specifies the line separator character. See "Line Separator Values" on page 115 for a list of commonly used values. This property is only used when converting HTML to XHTML. The default is crlf.
warnings	Abort on warnings found	Determines whether the converter will fail if it encounters problems with the HTML-to-XHTML mapping. Valid values: n yes – the converter fails when it encounters problems with the mapping. n no – even if the converter encounters problems with the mapping, the converter continues with the conversion, making a best guess. The default is no.

Java .properties File XML Converter Properties

You can use the JavaProps XML Converter to convert Java .properties files to XML and vice versa.

The following table lists the properties for the JavaProps XML Converter.

XML Converter Name in URI

JavaProps

Table 5-17. Properties for the JavaProps XML Converter

Name in URI	Property Name	Description
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is ISO-8859-1.
newline	Line separator	Specifies the line separator character. See "Line Separator Values" on page 115 for a list of commonly used values. This property is used when converting a file to XML, and vice versa. The default is crlf.

JSON XML Converter Properties

The following table lists properties for JSON (JavaScript Object Notation) Stylus XML Converters.

XML Converter Name in URI

JSON

Table 5-18. Properties for the JSON XML Converter

Name in URI	Property Name	Description
indent	Indent	Specifies the level of indent to use for the converted XML. The default is 4.
newline	Line separator	Specifies the line separator character. See "Line Separator Values" on page 115 for a list of commonly used values. The default is crlf.

OpenEdge .d Data Dump XML Converter Properties

You can use the DotD XML Converter to convert Progress OpenEdge .d data dump files to XML and vice versa.

The following table lists properties for the DotD XML Converter.

XML Converter Name in URI

DotD

Table 5-19. Properties for the DotD XML Converter

Name in URI	Property Name	Description
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is utf-8.
newline	Line separator	Specifies the line separator character. See "Line Separator Values" on page 115 for a list of commonly used values. This property is used when converting a file to XML, and vice versa. The default is crlf.

Pyx Format XML Converter Properties

You can use the Pyx XML Converter to convert Pyx format files to XML and vice versa.

The following table lists the properties for the Pyx XML Converter.

XML Converter Name in URI

Pyx

Table 5-20. Properties for the Pyx XML Converter

Name in URI	Property Name	Description
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is utf-8.
newline	Line separator	Specifies the line separator character. See "Line Separator Values" on page 115 for a list of commonly used values. This property is used when converting a file to XML, and vice versa. The default is crlf.

Rich Text Format XML Converter Properties

The following table lists the properties for the Rich Text Format (RTF) XML Converter.

XML Converter Name in URI

RTF

Table 5-21. Properties for the RTF XML Converter

Name in URI	Property Name	Description
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is cp850.
newline	Line separator	Specifies the line separator character. See “Line Separator Values” on page 115 for a list of commonly used values. This property is used when converting a file to XML, and vice versa. The default is crlf.

SDI XML Converter Properties

You can use the SDI XML Converter to convert Super Data Interchange Format (SDI) files to XML and vice versa.

The following table lists the properties for the SDI XML Converter.

XML Converter Name in URI

SDI

Table 5-22. Properties for the SDI XML Converter

Name in URI	Property Name	Description
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is cp850.
newline	Line separator	Specifies the line separator character. See "Line Separator Values" on page 115 for a list of commonly used values. This property is used when converting SDI files to XML, and vice versa. The default is crlf. S.

SYLK XML Converter Properties

You can use the SYLK XML Converter to convert Symbolic Link Format (SYLK) files to XML and vice versa.

The following table lists the properties for the SYLK XML Converter.

XML Converter Name in URI

SYLK

Table 5-23. Properties for the SYLK XML Converter

Name in URI	Property Name	Description
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is cp850.
newline	Line separator	Specifies the line separator character. See "Line Separator Values" on page 115 for a list of commonly used values. This property is used when converting SYLK files to XML, and vice versa. The default is crlf.

Tab-Separated Values XML Converter Properties

You can use the TAB XML Converter to convert tab-separated values files to XML and vice versa.

The following table lists the properties for the TAB XML Converter.

XML Converter Name in URI

TAB

Table 5-24. Properties for the Tab-Separated Values XML Converter

Name in URI	Property Name	Description
collapse	Collapse consecutive separators	Determines whether to collapse consecutive separators, separators that do not contain any data. Valid values: n yes – consecutive separators are collapsed. n no – consecutive separators are not collapsed. The default is no.
double	Doubling embedded quote escapes it	Determines whether doubling an embedded quotation mark has the effect of escaping the quoted string. Valid values: n yes – allows doubling quotation marks to escape a quoted string. n no – does not permit doubling quotation marks to escape a quoted string. The default is no.

Table 5-24. Properties for the Tab-Separated Values XML Converter

Name in URI	Property Name	Description
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is cp1252.
escape	Escape character	Specifies the escape character (escapes quotes and separators so that they can be embedded in values). The backslash (\) is the default.
first	First row contains field names	Generated field names depend on the values in the first and number fields. If first=yes and number=no, field names are read from the first row. Any field names after that are named column.*nnn*, where *nnn* is the column number, starting from 1 and including explicitly named columns in the count. If number=yes, extra columns (those after the first) are named column.
multiline=	Multiline	Determines whether a line separator in a quoted string is considered part of the content of a field. Valid values: n yes – a line separator in a quoted string is considered part of the content of that field. n no – a line separator in a quoted string is not considered part of the content of a field and terminates the row, even if it is encountered in the middle of a quoted string. The default is no.
newline	Line separator	See “Line Separator Values” on page 115 for a list of commonly used values.

Table 5-24. Properties for the Tab-Separated Values XML Converter

Name in URI	Property Name	Description
number	Number rows and columns	Determines whether rows are numbered. Valid values: yes – each row has an attribute named row that specifies the row number from the source document, starting from 1. Also, each column, even those explicitly named, have a column attribute numbering the column from 1. no – any empty columns are omitted from the output, but the numbering of subsequent columns reflect that a column(s) was skipped. The default is no.
quotes	Quote characters	Specifies a list of characters that the converter should interpret as quotation characters. The default is double quotes (") and single quotes (’).
root	Root element name	Specifies the root element name. The default is table.
row	Row element name	Specifies the row element name. The default is row.
sep	Separator	Specifies the separator value between each value. This can be TAB, any single character (a comma (,) is the default), or the %*XX*-escaped value (%2c, for example).

Whole-Line Text XML Converter Properties

You can use the Line XML Converter to convert whole-line text formatted files to XML and vice versa.

The following table lists the properties for the Line XML Converter.

XML Converter Name in URI

Line

Table 5-25. Properties for the Whole-line Text XML Converter

Name in URI	Property Name	Description
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is utf-8.
line	Line element name	Specifies the line element name. The default is line.
newline	Line separator	Specifies the line separator character. See “Line Separator Values” on page 115 for a list of commonly used values. This property is used when converting a whole-line text file to XML, and vice versa. The default is crlf.
root	Root element name	Specifies the root element name. The default is root.

Windows .ini File XML Converter Properties

You can use the WinIni XML Converter to convert Windows .ini files to XML and vice versa.

The following table lists the properties for the WinIni XML Converter.

XML Converter Name in URI

WinIni

Table 5-26. Properties for the WinIni XML Converter

Name in URI	Property Name	Description
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is cp1252.
newline	Line separator	Specifies the line separator character. See "Line Separator Values" on page 115 for a list of commonly used values. This property is used when converting a file to XML, and vice versa. The default is crlf.

Windows Write XML Converter Properties

You can use the WinWrite XML Converter to convert Windows Write files to XML and vice versa.

The following table lists the properties for the Windows Write XML Converter.

XML Converter Name in URI

WinWrite

Table 5-27. Properties for the WinWrite XML Converter

Name in URI	Property Name	Description
encoding	Encoding	Specifies the encoding for the input file when the input file is not XML or the encoding for the output file when the output file is not XML. The default is utf-8.
newline	Line separator	Specifies the line separator character. See "Line Separator Values" on page 115 for a list of commonly used values. This property is used when converting a file to XML, and vice versa. The default is crlf.

Index

A

B

C

D

E

F

G

H

I

J

L

N

O

P

R

S

T

U

W

X