



DataDirect **XML Converters™**

User's Guide and Reference
for Java

Release 4.0
December 2008

© 2008 Progress Software Corporation. All rights reserved.

Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. This manual is also copyrighted and all rights are reserved. This manual may not, in whole or in part, be copied, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Progress Software Corporation.

The information in this manual is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear in this document. The references in this manual to specific platforms supported are subject to change.

A (and design), Actional, Actional (and design), Affinities Server, Allegrix, Allegrix (and design), Apama, Business Empowerment, DataDirect (and design), DataDirect Connect, DataDirect Connect64, DataDirect Connect OLE DB, DataDirect Technologies, DataDirect XQuery, DirectAlert, Dynamic Index Utility, Dynamic Routing Architecture, EasyAsk, EdgeXtend, Empowerment Center, eXcelon, Fathom, Halo, IntelliStream, Iwave Integrator, LiveMine, Neon, Neon 24x7, Neon New Era of Networks, Neon Unload, O (and design), ObjectStore, OpenEdge, PDF, PeerDirect, Persistence, Persistence (and design), POSSENET, Powered by Progress, PowerTier, ProCare, Progress, Progress Dynamics, Progress Business Empowerment, Progress Empowerment Center, Progress Empowerment Program, Progress Fast Track, Progress OpenEdge, Progress Profiles, Progress Results, Progress Software Developers Network, ProtoSpeed, ProVision, PS Select, SequeLink, Shadow, ShadowDirect, Shadow Interface, Shadow Web Interface, Shadow Web Server, Shadow TLS, SOAPStation, Sonic ESB, SonicMQ, Sonic Orchestration Server, Sonic Software (and design), SonicSynergy, Speed Load, SpeedScript, Speed Unload, Stylus Studio, Technical Empowerment, UIM/X, Visual Edge, Voice of Experience, WebSpeed, and Your Software, Our Technology-Experience the Connection are registered trademarks of Progress Software Corporation or one of its subsidiaries or affiliates in the U.S. and/or other countries. AccelEvent, A Data Center of Your Very Own, Apama Dashboard Studio, Apama Event Manager, Apama Event Modeler, Apama Event Store, AppsAlive, AppServer, ASPen, ASP-in-a-Box, BusinessEdge, Cache-Forward, Connect Everything, Achieve Anything., DataDirect Spy, DataDirect SupportLink, DataDirect Test, DataDirect XML Converters, DataXtend, Future Proof, Ghost Agents, GVAC, Looking Glass, ObjectCache, ObjectStore Inspector, ObjectStore Performance Expert, Pantero, POSSE, ProDataSet, Progress DataXtend, Progress ESP Event Manager, Progress ESP Event Modeler, Progress Event Engine, Progress RFID, PSE Pro, PS Select, SectorAlliance, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Sonic Business Integration Suite, Sonic Process Manager, Sonic Collaboration Server, Sonic Continuous Availability Architecture, Sonic Database Service, Sonic Workbench, Sonic Orchestration Server, Sonic XML Server, WebClient, and Who Makes Progress are trademarks or service marks of Progress Software Corporation or one of its subsidiaries or affiliates in the U.S. and other countries.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Any other trademarks or service marks contained herein are the property of their respective owners.

Stylus Studio includes:

Xerces c++ developed by the Apache Software Foundation ([http:// www.apache.org/](http://www.apache.org/)). Copyright © 1999-2006 the Apache Software Foundation. All rights reserved.

XercesJ developed by the Apache Software Foundation ([http:// www.apache.org/](http://www.apache.org/)). Copyright © 1999-2006 the Apache Software Foundation. All rights reserved.

FOP developed by the Apache Software Foundation ([http:// www.apache.org/](http://www.apache.org/)). Copyright © 1999-2006 the Apache Software Foundation. All rights reserved.

Axis developed by the Apache Software Foundation ([http:// www.apache.org/](http://www.apache.org/)). Copyright © 1999-2006 the Apache Software Foundation. All rights reserved.

The names "Xalan", "FOP", and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation. For written permission, please contact apache@apache.org.

Files that are subject to the DSTC Public License (DPL) Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at [http:// www.dstc.com](http://www.dstc.com). Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is xs3p. The Initial Developer of the Original Code is DSTC. Portions created by DSTC are Copyright © 2002. All rights reserved.

Pathan developed by DecisionSoft Limited. Copyright © 2001 DecisionSoft Limited. All rights reserved.

Software developed by Thai Open Source Software Center Ltd. Copyright © 2001-2003, Thai Open Source Software Center Ltd. All rights reserved.

IBM ICU developed by IBM. Copyright © 1995-2003 International Business Machines Corporation and others. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

Software developed by Kevin Atkinson. Copyright © 2000-2004, by Kevin Atkinson. All rights reserved.

Aspell 0.60.2, from the Free Software Foundation, Inc. (<http://www.fsf.org/>), which is subject to the GNU Lesser General Public License Version 2.1 (<http://www.gnu.org/licenses/lgpl.html>). Software distributed under this license is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license.

Software developed by xqDoc.org. Copyright © 2005 Elsevier, Inc. All rights reserved.

Software developed by Info-ZIP. Copyright © 1990-2004 Info-ZIP. All rights reserved. For the purposes of this copyright and license, "Info-ZIP" is defined as the following set of individuals: Mark Adler, John Bush, Karl Davis, Harald Denker, Jean-Michel Dubois, Jean-loup Gailly, Hunter Goatley, Ian Gorman, Chris Herborth, Dirk Haase, Greg Hartwig, Robert Heath, Jonathan Hudson, Paul Kienitz, David Kirschbaum, Johnny Lee, Onno van der Linden, Igor Mandrichenko, Steve P. Miller, Sergio Monesi, Keith Owens, George Petrov, Greg Roelofs, Kai Uwe Rommel, Steve Salisbury, Dave Smith, Christian Spieler, Antoine Verheijen, Paul von Behren, Rich Wales, Mike White. Info-ZIP software is provided "as is", without warranty of any kind, express or implied. In no event shall Info-ZIP or its contributors be held liable for any direct, indirect, incidental, special or consequential damages arising out of the use of or inability to use this software.

Software developed by Tim Bray and Sun Microsystems and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. Copyright © 2004 Tim Bray and Sun Microsystems. All rights reserved.

Software developed by Saxonica Limited and is distributed on an "AS IS" basis WITHOUT WARRANTY OF

ANY KIND, either express or implied. Copyright © 2005 Saxonica Limited. All rights reserved.

Software developed by the The Anti-Grain Geometry Project. Copyright © 2002-2005 Maxim Shemanarev (McSeem). This software is provided "as is" without express or implied warranty, and with no claim as to its suitability for any purpose.

DataDirect XQuery

DataDirect XML Converters

DataDirect XML Converters include:

JavaMail and JavaBeans Activation Framework software developed by Sun Microsystems. Copyright © 1994-2006, Sun Microsystems, Inc. All rights reserved.

Software developed by World Wide Web Consortium. Copyright (c) 1998-2003 World Wide Web Consortium (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved.

Software developed by World Wide Web Consortium. Copyright (c) 1998-2000 World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

Software developed by JSON.org. Copyright (c) 2002 JSON.org. All rights reserved.

December 2008

Table of Contents

Preface	9
What are DataDirect XML Converters™?	9
Using This Book	10
Typographical Conventions	11
Contacting Technical Support	12
1 DataDirect XML Converters™ Overview	15
Types of XML Converters	15
XML Converters Can Be Customized	17
Data Access	18
URI Schemes	19
Command Line Usage	19
Usage Notes	20
Example	21
Handling Proprietary EDI Formats	21
Creating a SEF File	21
The SEF Specification	22
Example: Using a SEF File	22
Error Handling Support	23
ConverterListener Interface	23
EDICConverterListener Interface	24
EDICConverterException Interface	24

XML Schema Generation	25
Command Line Usage	26
Example Scenario	26
Instance Documents	28
URI Parameters That Affect XML Schema	29
XML Schema Generation Summary	32
Example Applications	34
Converting EDI to XML	34
Creating XML Schemas from EDI	35
2 XML Converters™ URI Schemes	37
The converter: URI Scheme	37
Converter URI Syntax	37
Example	38
Specifying XML Converter Properties	39
Building a converter: URI	39
Where Converter URIs are Displayed in Stylus Studio	40
Invoking a Custom XML Conversion	41
Invoking a Converter URI in DataDirect XQuery®	42
More about DataDirect XQuery	42
3 XML Converters™ Examples	43
Overview of the demo.java Example	43
Demonstration Files	44

Running demo.java	45
Before You Begin	45
How to Run the Demonstration	45
Example 1	46
Example 2	47
Example 3	48
Example 4	49
Example 5	51
Example 6	52
Example 7	53
Example 8	54
Example 9	57
Example 10	58
Example 11	59
Processing Conversion Results	61
Loading SEF Files Programmatically	62
Using SEF Files Created with Stylus Studio	62
Using a SEF File for Multiple Conversions	63
4 XML Converters™ Properties	65
Line Separator Values	66
Base-64 XML Converter Properties	67
XML Converter Name in URL	67
Binary XML Converter Properties	68
XML Converter Name in URL	68
Comma-Separated Values (CSV) XML Converter Properties ..	69
XML Converter Name in URL	69
dBase XML Converter Properties	71
XML Converter Names in URL	71
Datatypes Supported by Version	72
DIF XML Converter Properties	73
XML Converter Name in URL	73

EDI XML Converter Properties	74
XML Converter Name in URL	74
Properties for EDI XML Converters	75
Using Special Characters for Separators	95
EDI Processing Instructions	100
HTML Converter Properties	102
XML Converter Name in URL	102
Java .properties File XML Converter Properties	103
XML Converter Name in URL	103
JSON XML Converter Properties	104
XML Converter Name in URL	104
OpenEdge .d Data Dump XML Converter Properties	105
XML Converter Name in URL	105
Pyx Format XML Converter Properties	106
XML Converter Name in URL	106
Rich Text Format XML Converter Properties	107
XML Converter Name in URL	107
SDI XML Converter Properties	108
XML Converter Name in URL	108
SYLK XML Converter Properties	109
XML Converter Name in URL	109
Tab-Separated Values XML Converter Properties	110
XML Converter Name in URL	110
Whole-line Text XML Converter Properties	112
XML Converter Name in URL	112
Windows .ini File XML Converter Properties	113
XML Converter Name in URL	113
Windows Write XML Converter Properties	114
XML Converter Name in URL	114
Index	115

Preface

This book is your guide and reference to DataDirect XML Converters™ from DataDirect Technologies and describes how to use DataDirect XML Converters to build Java applications that provide bi-directional access to non-XML data. This book provides information about the following topics:

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

What are DataDirect XML Converters™?

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

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

Using This Book

This manual describes DataDirect XML Converters and how to use them to develop Java applications. It is assumed that you are familiar with XML, Java, and related technologies.

This manual has the following chapters:

- [Chapter 1, “DataDirect XML Converters™ Overview”](#) provides an overview of the DataDirect XML Converters API and URI schemes used for data integration.
- [Chapter 2, “XML Converters™ URI Schemes”](#) describes the converter: URI scheme and how to use Stylus Studio XML Enterprise Suite to build converter: URIs.
- [Chapter 3, “XML Converters™ Examples”](#) describes `demo.java`, a simple Java program installed with the 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 4, “XML Converters™ Properties”](#) describes values for the properties for XML Converters.

Typographical Conventions

This book uses the following typographical conventions:

Convention	Explanation
<i>italics</i>	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.
<i>monospaced italics</i>	Indicates names that are placeholders for values you specify; for example, <i>filename</i> .
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.
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.

Contacting Technical Support

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

<http://support.datadirect.com>

The DataDirect Technologies Web site provides the latest support information through our global service network. The SupportLink 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 Knowledgebase for technical bulletins and other information.

To obtain technical support for an evaluation copy of the product, go to:

http://www.datadirect.com/support/eval_help/index.ssp

or contact your sales representative.

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

- 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. If you do not have a SupportLink contract, the SupportLink representative assisting you will connect you with our Sales team.
- Your name, phone number, email address, and organization. For a first-time call, you may be asked for full customer information, including location.
- The DataDirect product and the version that you are using.
- The type and version of the operating system where you have installed your DataDirect product.

- Any EDI, flat file, legacy file, custom XML conversion definition, 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 recreated.
- A description of what you have attempted to resolve the issue. If you have researched your issue on Web search engines, our Knowledgebase, 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.

1 DataDirect XML Converters™ Overview

DataDirect 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 as XML through Java applications.

This chapter provides an overview of the XML Converters, including the types of file formats they support, 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 XML Converters

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

The following table summarizes the types of available XML Converters.

Table 1-1. File Formats Supported by XML Converters

File Type	Description	Bidirectional
ATIS, EANCOM, EDIFACT, Edig@s, HIPAA, HL7, IATA, and X12	Automatically detects and parses ATIS, EANCOM, EDIFACT, Edig@s, HIPAA, HL7, IATA, and X12 EDI message types, with options for custom message types and message extensions to cover proprietary EDI-based formats.	Yes
Base-64	Converts any file, text or binary (such as an image), into a 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 or octal or even binary.	Yes
CSV	Converter for comma-separated values (CSV) files. Supports multiple encodings and options to tune the quote and escape characters. Supports delimiters besides 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 Progress Software's OpenEdge text dump file format.	Yes
HTML	Support for HTML to XML (XHTML) file conversion.	Yes
JavaProps	Support for 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

Table 1-1. File Formats Supported by XML Converters

File Type	Description	Bidirectional
Line	Reads in text one line at a time, wrapping an element around each line and escaping any embedded & or > or < symbols.	Yes
Pyx	Support for this 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 XML Converters for DIF and SYLK.	Yes
SYLK	SYLK (Symbolic Link) is another popular spreadsheet-based file format. There are also XML Converters for DIF and SDI.	Yes
TAB	Tab-separated values format commonly associated with MS Excel spreadsheets.	Yes
WinIni	Converter for Windows .ini configuration files.	Yes
WinWrite	Converter for Microsoft WinWrite files; renders XHTML.	No
Custom	Custom XML conversions (.conv files) created using Stylus Studio.	No

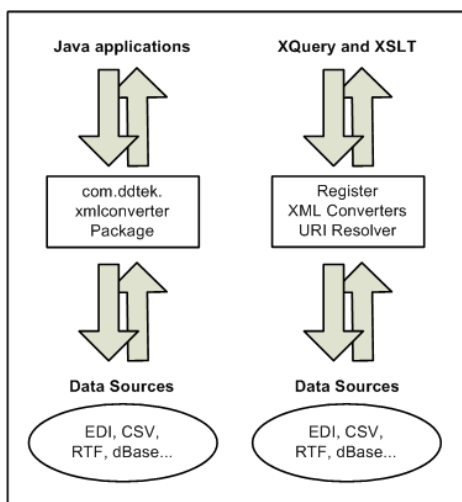
XML Converters Can Be Customized

Every XML Converter has properties that allow you to tailor the converter to suit your needs. Some XML 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 XML Converter.

See [“URI Schemes” on page 19](#) to learn more about the converter: URI. See [Chapter 4 “XML Converters™ Properties” on page 65](#) for more information.

Data Access

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



Data access to non-XML data stored as EDI or other file formats (CSV, tab-delimited, and others, for example) is accomplished using the DataDirect XML Converters converter: URI scheme. See [“URI Schemes” on page 19](#) to learn more about the URI schemes supported by XML Converters.

URI Schemes

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

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

For example, the URI scheme `converter:myConverter.conv` invokes the custom XML conversion `myConverter.conv` file. The URI scheme `converter:EDI?file:///m:/testing/editeur.edi` invokes the XML Converters EDI converter, using the file `editeur.edi` as the EDI source to be converted.

See [Chapter 2 “XML Converters™ URI Schemes”](#) for more information.

Command Line Usage

You can run DataDirect XML Converters™ from the command line.

To specify a native file to be converted to XML:

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

Usage Notes

Following are some usage notes for running DataDirect XML Converters™ from the command line:

- The XML Converter specified in the `<name>` argument for the `converter` option can take 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, `newline = platform` is not.
- Use `/in -` to read from the standard input.
- To write to the standard output, omit the `/out` parameter.
- You can use dashes (-) instead of forward slashes (/) to separate parameters – for example, `-to` instead of `/to`.
- To generate XML Schema, replace the `/to` option with `/schema`. See [“XML Schema Generation” on page 25](#) 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:

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

The 831.x12 sample file and others are in the \examples directory where you installed XML Converters. To learn more about the examples, see [“Example Applications” on page 34](#).

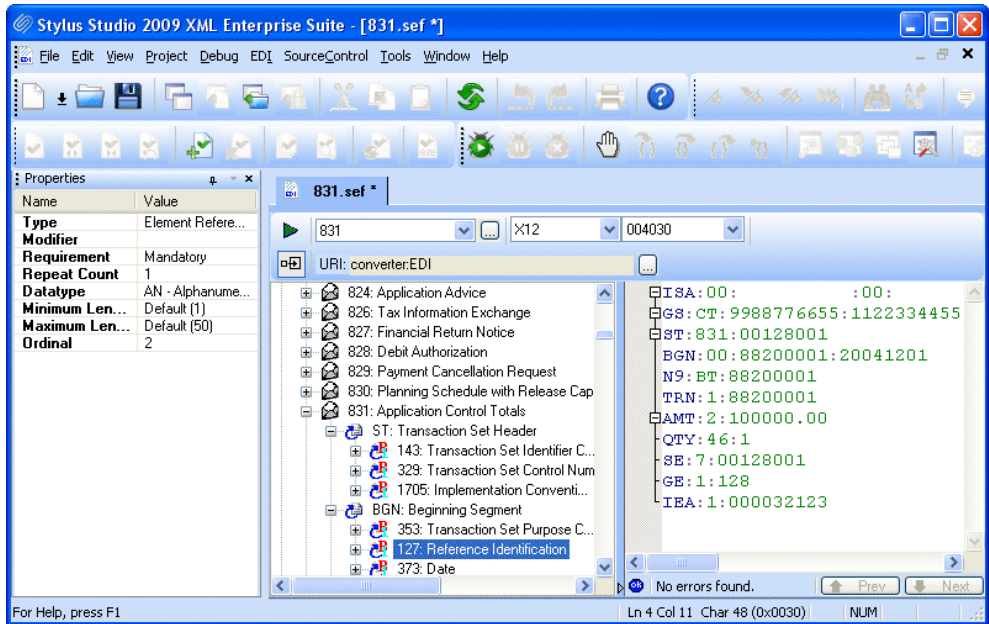
Handling Proprietary EDI Formats

DataDirect 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 like EDIFACT or X12, for example. You save this structure definition in a SEF file, which you can then instruct the EDI XML Converter to use when converting proprietary EDI to XML.

Creating a 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 EDI to XML Module, which provides a visual editor to help you build SEF files based on standard EDI dialects.

When using the 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 a copy of the SEF specification on the DataDirect XML Converters web site:

<http://www.xmlconverters.com/SEF/sef161.pdf>

You need Adobe Acrobat Reader Version 4.0 or later to view this document. You can download the free Adobe Acrobat Reader [here](#).

Example: Using a SEF File

See “[Example 7](#)” on page 53 for an example of using an SEF extension file to define an XML Schema. See also “[Loading SEF Files Programmatically](#)” on page 62.

Error Handling Support

The DataDirect XML Converters™ API provides several ways to manage error handling in your applications:

- ConverterListener interface
- EDIConverterListener interface
- EDIConverterException class

These interfaces are currently implemented only for the EDI XML Converter.

ConverterListener Interface

Warnings and errors do not always need to throw exceptions and abort a conversion process; in these cases it can be desirable 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 `ConverterListener` 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; by simply not throwing the exception received, processing continues. In the case of an error, it is possible that other errors will cascade from the first. Fatal errors can be reported, but upon return a `ConverterException` is always thrown by the EDI XML Converters™ engine. The exception that is thrown will be an instance of `ConverterException` or one of its subclasses such as `ConverterArgumentException`.

Example

See [“Example 8” on page 54](#) for an example of registering a `ConverterListener`.

EDIConverterListener Interface

The `EDIConverterListener` is a specialized version of `ConverterListener`; its methods provides more 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; `unknownCodeListValue()` is called when codelist validation fails.

EDIConverterException Interface

EDI-based conversions are typically 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, EDI XML Converters™ allow you to capture the error and possibly return standard EDI messages back to the sender of the message – `CONTRL` (for EDIFACT), `997` (for X12), or `ACK` (for HL7), for example.

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, in most cases the exception thrown will be `EDIConverterException`.

This exception contains many methods to probe the context of the specific error – `getContentData()`, `getControlData()`, `getData()`, and `getError()`. Processing can recover from both

warnings and errors; fatal errors, however, always end the processing.

Error Diagnostics

Full context information is provided when an error is encountered within 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 the error occurring matches one of those, it is reported as such.

XML Schema Generation

You can use the `SchemaGenerator` interface to create XML Schema files that describe the structure of XML files read or created by a `ConvertToXML` or `ConvertFromXML` object. You might 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. (This functionality is available only with certain XML Converters. See [“XML Schema Generation Summary” on page 32](#) for more information.)

The XML Schema of the generated file depends on the type of file for which the XML Schema is being created, and not on the actual data. For example, if you are creating an XML Schema for an EDI file, the 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 28](#) and [“URI Parameters That Affect XML Schema” on page 29](#) for more information.

Command Line Usage

To generate XML Schema from the command line:

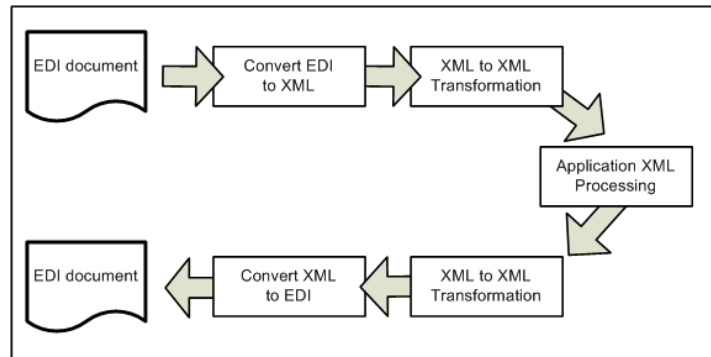
```
java -jar xmlconverters.jar /schema /converter name  
[:property_name=value [ :property_name=value ...] ] /in filename  
[/out filename]
```

Note that some XML Converters (like EDI, for example) require that you either provide an instance document or that you specify sufficient properties in the converter: URI. For EDI, for example, you need to provide dialect, version, and message type/transaction set.

Example Scenario

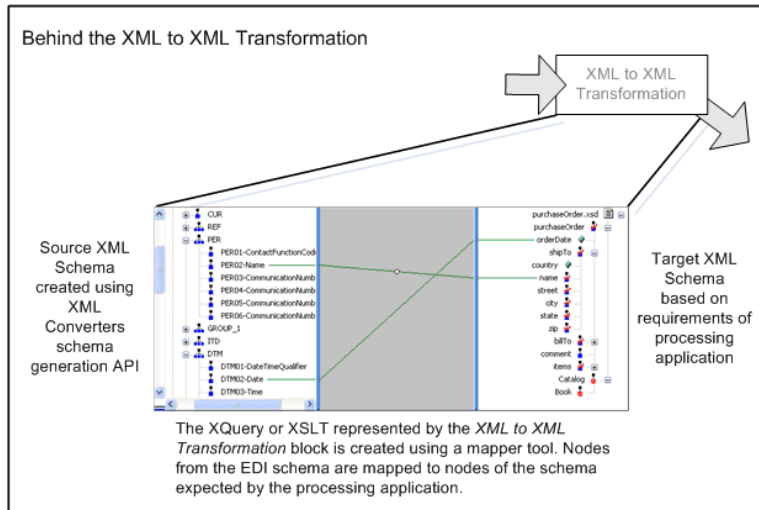
Consider the following example scenario: your enterprise routinely receives client data in EDI files. The data in these files needs to be converted to XML so that it can be transformed for processing by an application. After application processing, the resulting XML is again transformed to another format before being converted back to EDI.

Such a workflow can be represented with the following diagram:



In this illustration, the conversion block *Convert EDI to XML* represents an instance of DataDirect XML Converters™, which is used to convert the incoming EDI document – say it’s 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. However, before the XML can be used by the processing application, it needs to be transformed into the format expected by the processing application – that is, it must conform to an XML Schema.

Imagine that XQuery is used to perform this transformation (the block *XML to XML Transformation* in the preceding illustration). One way to create such an XQuery is to use a mapping tool. In this case, we 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. Here, we use DataDirect XML Converters™ 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), this time 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 DataDirect XML Converters™.

See [“Creating XML Schemas from EDI”](#) on page 35 for an example of using DataDirect XML Converters™ to create XML Schema from an EDI document.

Instance Documents

Some XML Converters, like those for CSV and Tab, require an instance document in order to provide DataDirect XML Converters™ with the information it needs to generate an XML Schema. Other XML Converters (like EDI) can use instance documents, but they are not required; for this type of file, you can 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 DataDirect XML Converters™ built-in settings for XML Schema generation for files of that type.

See [“XML Schema Generation Summary” on page 32](#) for more information concerning instance document and converter: URI property usage for XML Converters.

URI Parameters That Affect XML Schema

This section describes how URI parameters affect XML Schema generation for the XML Converters that support their use. The XML Converters for which you can specify URI parameters are:

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

CSV XML Converter URI Parameters

The following parameters affect XML Schema generation for both CSV and tab-delimited files:

Table 1-2. Parameters for 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 converted XML; also specifies the name of the root element in generated XML Schema.
row=	Specifies the name of the row element in converted XML; also specifies the name of the row element in generated XML Schema.

EDI XML Converter URI Parameters

The following parameters affect XML Schema generation for EDI files:

Table 1-3. Parameters for EDI XML Converters

Property	Description
dialect=	Must be one of the following: ATIS, EANCOM, EDIFACT, EDIGAS, HIPAA, HL7, IATA, or X12. Required if an instance document is not provided.
doc=	Whether or not to include <code>xs:documentation</code> comments in the XML Schema. Default is "yes."

Table 1-3. Parameters for EDI XML Converters

Property	Description
inter=	Certain EDI messages have alternate batch and interactive forms, depending upon whether they are used between systems that have real-time connections. The <code>inter=yes</code> setting causes the interactive form to be 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.
long=	Whether you want to use long or short element names in your XML conversions – FTX03-TextReference (<code>long=yes</code>) or FTX03 (<code>long=no</code>), for example.
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 include SPORES and TKTRES; and so on. Required if an instance document is not provided.
tbl=	Whether or not the codelist tables are created as enumerations in the generated XSD output. Default is "no."
user=	Optionally specifies a SEF extension file, whose structure 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 include 991 and 992; and so on. Required if an instance document is not provided.

Line XML Converter URI Parameters

The following parameters affect XML Schema generation for whole-line text files:

Table 1-4. Parameters for Line XML Converters

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

Tab XML Converter URI Parameters

See [“CSV XML Converter URI Parameters”](#) on page 30.

XML Schema Generation Summary

The following table summarizes information about the XML Schema generation capabilities of DataDirect XML Converters. Note that not all XML Converters can generate XML Schema.

Table 1-5. XML Converters That Can Generate XML Schema

<i>Converter Name</i>	<i>Can Generate XML Schema</i>	<i>Instance Document</i>	<i>URI Parameters Affect XML Schema</i>
Base64	Yes	Not needed	No
Binary	Yes	Not needed	No

Table 1-5. XML Converters That Can Generate XML Schema

Converter Name	Can Generate XML Schema	Instance Document	URI Parameters Affect XML Schema
CSV	Yes	Mandatory	Yes
<i>custom</i>	Yes	Not needed	via .conv file
dBase (all)	Yes	Mandatory	No
DIF	Yes	Not needed	No
DotD	Yes	Not needed	No
EDI (all)	Yes	Optional	Yes
HTML	No	n/a	n/a
JavaProps	Yes	Not needed	No
JSON	No	n/a	n/a
Line	Yes	Not needed	Yes
Pyx	No	n/a	n/a
RTF	No	n/a	n/a
SDI	Yes	Not needed	No
Sylk	Yes	Not needed	No
Tab	Yes	Mandatory	Yes
WinIni	Yes	Not needed	No
WinWrite	Yes	Not needed	No

Example Applications

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

See [Chapter 3 “XML Converters™ Examples” on page 43](#) for other application examples.

Converting EDI to XML

Following 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 follows:

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

Creating XML Schemas from EDI

Following is a simple example that shows a Java program that generates 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=yes: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();
        }
    }
}
```


2 XML Converters™ URI Schemes

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

The converter: URI Scheme

The converter: URI scheme specifies an XML Converter name – either one of the standard XML Converters (for EDI or tab-delimited files, for example) and settings for the properties you wish to use, or 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 parameter of the converter: URI.

Converter URI Syntax

While 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 ]
```

To specify a converter: URI, you need to identify:

- The XML Converter you want to use (EDI, CSV, dBase, and so on)
- Options for that XML Converter (separator and escape characters, for example)

- The file to be converted

The format of the converter: URI string

Example

This converter: URI invokes the XML Converter for comma-separated values to convert the `three.txt` file in the `\XMLConverters\examples\` directory to XML:

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

The instructions to the XML Converter engine from this instance of the converter URI are described the following table.

Instruction	Converter URI String
Use the Comma-Separated Values XML Converter converter	<code>converter:CSV</code>
The line separator in the source file is a line feed	<code>newline=lf</code>
The values in the first row of the source file should be used to supply field names	<code>first=yes</code>
The source file is <code>three.txt</code>	<code>?file:///c:/XMLConverters/examples/three.txt</code>

In this example:

- The name of the XML Converter is CSV. It could be any XML Converter – EDI, Base64, DIF, RTF, and so on.
- Only the `newline=` and `first=` properties have been specified; default values are used for all other converter properties.
- The source file being converted is `three.txt` on `c:/XMLConverters/examples`. If you are using the converter: URI programmatically, you omit the `?URI` parameter as the source file being converted is specified by the program.

Specifying XML Converter Properties

XML Converter properties that use default values do not have to be specified in the URI. A comma is the default separator character for the CSV XML Converter, for example. But if the particular file you were converting used another separator character, you would need to specify it using the `sep=` property.

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

For a complete description of properties for all XML Converters, see [Chapter 4 “XML Converters™ Properties”](#).

Building a converter: URI

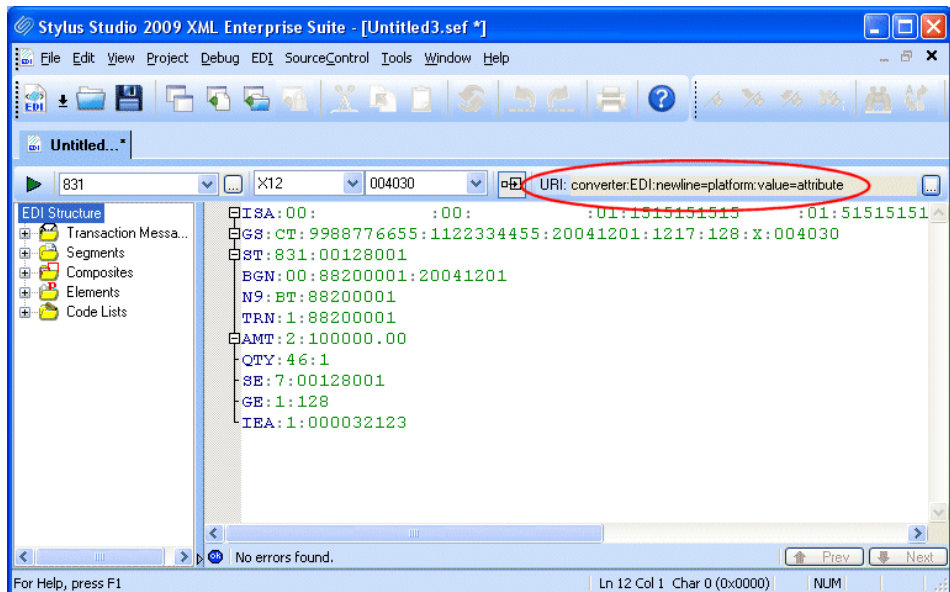
If you have Stylus Studio XML Enterprise Suite, you can use Stylus Studio to build the converter URIs you use in your Java applications. Converter URIs can be long and complex – properties and their values vary from one converter to another, for example – so using Stylus Studio to construct them can reduce errors in your applications. Its graphic user 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. For a complete description of properties for all XML Converters, see [Chapter 4 “XML Converters™ Properties”](#).

Where Converter URIs are Displayed in Stylus Studio

If you are using Stylus Studio XML Enterprise Suite, you can view converter URIs here:

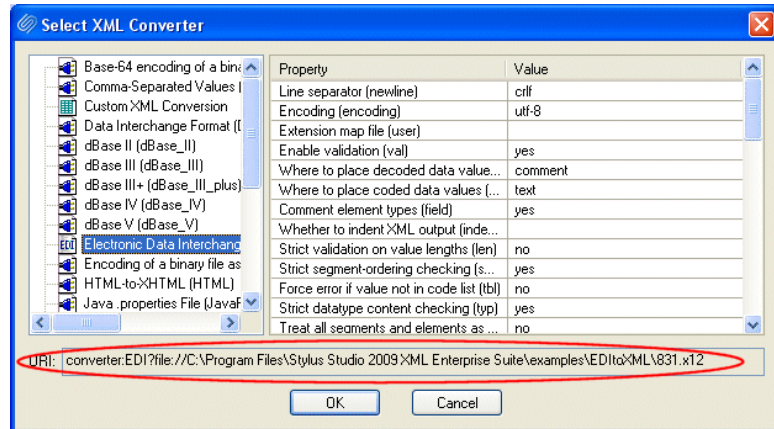
- In the **URI** field of the EDI to XML Module editor:



- 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 either source for the converter: URI strings in your Java applications. For more information, see the Stylus Studio product documentation.

Invoking a Custom XML Conversion

The converter: URI scheme can also be used to reference a custom XML conversion (a .conv file) built using Stylus Studio XML Enterprise Suite. In this case, 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.

A converter URI that references a custom XML conversion called myConverter.conv might look like this:

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

This converter: URI uses myConverter.conv to convert the file inventory.txt to some format (specified in the custom XML

conversion when you built it using Stylus Studio XML Enterprise Suite).

NOTE: Custom XML conversions can be defined *only* using Stylus Studio XML Enterprise Suite.

Invoking a Converter URI in DataDirect XQuery®

DataDirect XQuery®, an XQuery implementation, uses a document URI resolver that enables the `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 Converters properties is set (`first=yes`); default settings are used for all other properties.

More about DataDirect XQuery

DataDirect 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 DataDirect XQuery, visit the DataDirect XQuery web site at <http://www.xquery.com>.

3 XML Converters™ Examples

The DataDirect 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 DataDirect XML Converters `\examples` folder that demonstrates some of the features of DataDirect XML Converters and the Converters API.

This chapter also provides examples of using DataDirect XML Converters that are not part of the `\examples` folder.

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 DataDirect 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.

Demonstration Files

The files required to run the demonstrations are summarized in the following table. All of these files are installed in the `\examples` directory where you installed the XML Converters.

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

Running demo.java

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

Before You Begin

In order to recompile the files in the demonstration application, ensure that JDK 1.4.x or later is installed on your machine, and that its \bin directory is included in your system's PATH.

How to Run the Demonstration

To start the demo.java demonstration:

- 1 Open a console window.
- 2 Change to the \examples directory where you installed the XML Converters.
- 3 Run demo.bat (or demo.sh, if you installed the XML Converters on a Unix 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 since we, and not the Converter object, opened them.

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";

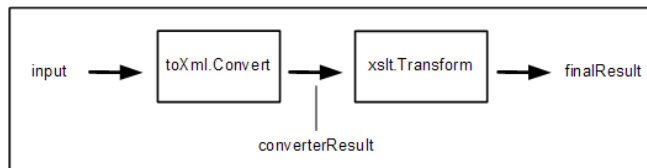
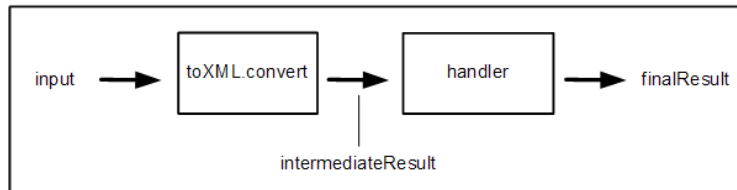
    Converter toXml = factory.newConvertToXML(customConversion);
    toXml.convert(converterSource, converterResult);

    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 simple conversion of one file type to another – some type of converter (either a DataDirect XML Converter or a user-defined custom XML conversion) was given an input and converted it to another format.

Example 4 generates conversion output (in this case, a CSV file, `one.csv`, converted to XML using the DataDirect XML Converter) as SAX events. 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);

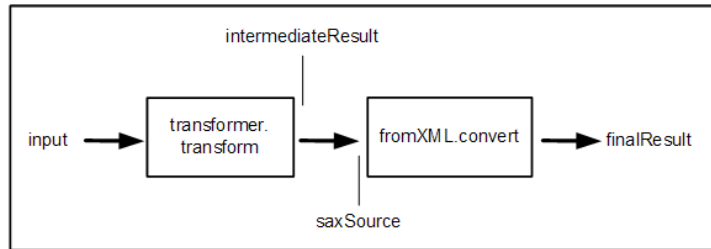
    xslt.transform(converterResult, xsltResult);

    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 permitted code value in the 353 element of segment BGN in transaction set 831.

The URL of the SEF file is specified in the `user=` parameter of the `converter: URL`. It is also possible to specify the SEF file name as a relative pathname. If XML Converters has been installed in a directory `PRODUCT_PATH`, and the EDI `converter: URL` contains `user=relative.sef`, then the SEF file will be found at `PRODUCT_PATH/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("test7 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 53](#), 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; note the error 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. Adjusting 'U' to '^'.
```

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. This fix has been automatically made to the data stream.

Converter error notification:

```
com.ddtek.xmlconverter.adapter.edi.EDIConverterException: [DDEE0008] ERROR
Value 99 not in codelist 353.
```

```
  Dialect: X12
  Version: syntax=00403/004030;message=00403/004030;agency=004030;table=
004030
  Message: 831
  Segment: BGN (line 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
```

Example 9

This example shows how to use the DataDirect XML Converters™ API to create an XML Schema based on a comma-separated values (CSV) file. Note that an instance document is required in order to generate XML Schema for CSV and other file types. See [“Instance Documents” on page 28](#) for more information.

The XML Schema generator is used very much like 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 25](#) 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 DataDirect XML Converters™ 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

- Using a sample EDI input (as shown in [“Example 9” on page 57](#)). See [“Instance Documents” on page 28](#) for more information.
- As part of the `converter: URL`, as demonstrated in this example. Also, see [“URI Parameters That Affect XML Schema” on page 29](#) for more information.

See [“XML Schema Generation” on page 25](#) for more information on this topic.

```
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 XML Converters 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 will read its input from "one.csv", convert it to XML, return 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 all at once, separating them with a '?' character 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);
}
}
}
}

```

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.

Sometimes it can be more efficient or desirable to 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. Following is its definition:

```
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()
```

`DataDirect 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.

XML Converters uses a byte stream if the source object contains one. If the source object does not contain a byte stream, XML Converters uses a character stream, if present. If neither a byte stream nor a character stream is available, XML Converters uses a `systemId`. If none of these is present in the source object, 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 `.PRIVATE` section. This converter: URI can contain XML Converters properties (`val=no` and `len=yes`, for example).

If you specify such a SEF file in your application, XML Converters uses the converter properties from that URI when performing the XML conversion. That is, XML Converter properties specified in the SEF become the new default values for the XML Converter 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().setEDIEExtension (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 `setEDIEExtension` 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().setEDIEExtension (StreamSource sef);
```


4 XML Converters™ Properties

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

This chapter provides reference information for the line separator property, which is common to most XML Converters, and reference information for individual XML Converters.

- [“Line Separator Values” on page 66](#)
- [“Base-64 XML Converter Properties” on page 67](#)
- [“Binary XML Converter Properties” on page 68](#)
- [“Comma-Separated Values \(CSV\) XML Converter Properties” on page 69](#)
- [“dBase XML Converter Properties” on page 71](#)
- [“DIF XML Converter Properties” on page 73](#)
- [“EDI XML Converter Properties” on page 74](#)
- [“HTML Converter Properties” on page 102](#)
- [“Java .properties File XML Converter Properties” on page 103](#)
- [“JSON XML Converter Properties” on page 104](#)
- [“OpenEdge .d Data Dump XML Converter Properties” on page 105](#)
- [“Pyx Format XML Converter Properties” on page 106](#)
- [“Rich Text Format XML Converter Properties” on page 107](#)
- [“SDI XML Converter Properties” on page 108](#)

- [“SYLK XML Converter Properties” on page 109](#)
- [“Tab-Separated Values XML Converter Properties” on page 110](#)
- [“Whole-line Text XML Converter Properties” on page 112](#)
- [“Windows .ini File XML Converter Properties” on page 113](#)
- [“Windows Write XML Converter Properties” on page 114](#)

Line Separator Values

Most XML Converters allow you to specify some type of line separator (referred to in the converter URI as *newline*). The following table summarizes commonly occurring values. All values are case-insensitive.

Table 4-1. Line Separator Values

Value	Description
cr or mac	The Macintosh standard.
crLf or dos	The DOS and Windows standard.
lf or unix	The Unix standard.
lfcr	Not standard usage.
nel	0x85 (commonly found in mainframes).
null	A null byte.
platform	If another value has not been specified, the line separator uses the platform value as returned by the <code>System.getProperty("line.separator")</code> method. platform is the default.

Base-64 XML Converter Properties

The following table shows XML Converters properties for Base-64 encoded binary files as documented in RFC 1341.

XML Converter Name in URL

Base-64

Table 4-2. Properties for Base-64 XML Converters

Name in URL	Property Name	Description
encoding	Encoding	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML. The default is utf-8.
newline	Line separator	Used only when converting a Base-64 binary file to XML, and not vice versa. The default is crlf. See "Line Separator Values" on page 66 for a list of values.

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 67 for more information.

XML Converter Name in URL

Binary

Table 4-3. Properties for Binary Base-2 to Base-36 XML Converters

Name in URL	Property Name	Description
base	Base	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	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML. The default is utf-8.
newline	Line separator	Used when converting a binary encoded file to XML, and vice versa. The default is crlf. See “Line Separator Values” on page 66 for a list of values.
space	Byte separator	Whether or not byte values should be 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	Whether you want to wrap lines (wrap=yes) or output all values on a single line (wrap=no).

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.

XML Converter Name in URL

CSV (comma-separated values)

Table 4-4. Properties for CSV XML Converters

Name in URL	Property Name	Description
collapse	Collapse consecutive separators	Whether or not you want to collapse consecutive separators – that is, separators that do not contain any data. Default is no.
double	Doubling embedded quote escapes it	Whether or not doubling an embedded quotation mark has the effect of escaping the quoted string. Default is no.
encoding	Encoding	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML.
escape	Escape character	This character escapes 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 <code>column.nnn</code> , where <code>nnn</code> is the column number, starting from one and including explicitly named columns in the count. If number=yes, extra columns (those after the first) are named just column.

Table 4-4. Properties for CSV XML Converters

Name in URL	Property Name	Description
newline	Line separator	See “Line Separator Values” on page 66 for a list of values.
number	Number rows and columns	<p>If number=yes (no is the default), each row will also have an attribute, named row, which will contain the row number from the source document, starting from one. Also, each column, even those explicitly named, will have a column attribute numbering the column from one.</p> <p>Any empty columns are omitted from the output, but the numbering of subsequent columns will reflect that a column(s) was skipped.</p>
quotes	Quote character	A list of characters the converter should interpret as quotation characters. Double and single quote marks (" ") are the default values.
root	Root element name	The value you want to use for the root element name. Default is table.
row	Row element name	The value you want to use for the row element name. Default is row.
sep	Separator	The separator value between each value. This can be TAB, any 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.

XML Converter Names in URL

- dBase_II
- dBase_III
- dBase_III_plus
- dBase_IV
- dBase_V

Table 4-5. Properties for dBase XML Converters

Name in URL	Property Name	Description
deleted	Include deleted records	Whether or not records marked with a "deleted" attribute are included in the output to XML and preserved in the conversion from XML. Stylus Studio generates the "deleted" attribute on output, and looks for it on input when this property is set to yes.
encoding	Encoding	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML. The default is utf-8.
newline	Line separator	Used only to convert a dBase file to XML, not vice versa. See " Line Separator Values " on page 66 for a list of values.

Datatypes Supported by Version

The following table identifies datatypes supported by dBase XML Converters.

Table 4-6. Datatype Support for dBase XML Converters

<i>Datatype</i>	<i>Symbol</i>	<i>dBase II</i>	<i>dBase III</i>	<i>dBase III+</i>	<i>dBase IV</i>	<i>dBase V</i>
binary	B					✓
character	C	✓	✓	✓	✓	✓
date	D		✓	✓	✓	✓
float	F				✓	✓
general	G					✓
logical	L	✓	✓	✓	✓	✓
memo	M		✓	✓	✓	✓
numeric	N	✓	✓	✓	✓	✓

DIF XML Converter Properties

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

XML Converter Name in URL

DIF (Data Interchange Format)

Table 4-7. Properties for the DIF XML Converter

<i>Name in URL</i>	<i>Property Name</i>	<i>Description</i>
encoding	Encoding	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML. The default is cp850.
newline	Line separator	Used when converting a DIF file to XML, and vice versa. The default is crlf. See “Line Separator Values” on page 66 for a list of values.

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 – ATIS, EANCOM, EDIFACT, Edig@s, HIPAA, HL7, IATA, and X12. Some properties are dialect-specific.

TIP: DataDirect XML Converters support the Standard Exchange Format (SEF) standard, which allows you to define extensions to an EDI standard. See [“Handling Proprietary EDI Formats” on page 21](#) for more information.

XML Converter Name in URL

EDI

Properties for EDI XML Converters

The following table describes properties for all EDI XML Converters. Note that some properties (cexpand and hexexpand, for example) are dialect-specific.

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
atis	Attempt to handle X12 as ATIS	Whether or not XML Converters should scan the first 100 segments of an X12 file to determine if it is an ATIS file. If the file is an ATIS file, ATIS rules are used; otherwise, X12 rules are used. Default is no.
auto	Auto-fixup values where possible	<p>Automatically calculates values where possible.</p> <p>For X12, it counts segments and fills in hash values for CTT, SE and IEA segments.</p> <p>For EDIFACT, it does segment totals for UNE, UNT, and UNZ segments. If the UNZ segment is missing, it is created as needed.</p> <p>The SE or UNT segments only need to be mapped; all of their elements can be automatically populated.</p> <p>In any header or trailer fields where there are dates or times, segments are placed into the correct format based on whether they are defined as YYMMDD or CCYYMMDD for dates, or whatever length for times. Default values will be filled in if they are missing.</p> <p>See also "count" on page 78.</p>
cent	Window for century cut-off	If the date is given in the file with a two-digit year and the output requires a four-digit year, this value is the cutoff so that the proper century can be selected.

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
cexpand	Fully expand HL7 CE/CF/CNE/CWE element	<p>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. If you set cexpand=yes, XML Converters attempts to expand all of the fields in the composite element.</p> <p>The CNE and CWE elements also allow pulling information from tables across versions of the standard – an HL7 2.4 element could look up information from an HL7 2.5 table, for example. Each of these also has an ‘alternate’ set of fields, so that codes can be included both in the native (HL7, for example) and foreign code list.</p>

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
chr	Character repertoire override	<p>Allows XML Converters to override and alter character encodings for EDIFACT-based documents (like EANCOM and IATA).</p> <p>You can use one or more of the following options, connected with a "+" symbol (chr=REPLACE+FINNISH, for example):</p> <ul style="list-style-type: none"> ■ DEFAULT – The encoding specified in the file is used. This option cannot be used with any others. ■ EANCOM – Support for these extra EANCOM characters to UNOA and UNOB are added: #, @, [,], {, }, \, , ', and ^. ■ SYMBOL – Forces all characters, including special characters such as element and segment separators that might otherwise be permitted, to be validated against the encoding. ■ REPLACE – Replaces any 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, XML Converters throws an error. ■ 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). <p>See "FINNISH Character Set Overrides" on page 92 for more information.</p> <p>See "Explicit Character Overrides" on page 93 for more information on this topic.</p>

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
clean	Remove linefeeds and nulls	Whether or not you want to remove linefeeds and nulls from EDI converted to XML and vice versa. Valid values are both (directions), fromXML, toXML, and never. See also "rtrim" on page 88.
component	Component value separator	When an element is a composite element, the character that is used to separate the component elements from each other within the composite element. See "Using Special Characters for Separators" on page 95 for information about how to specify values for this property.
count	Enforce segment maximum counts	Whether or not you want XML Converters to enforce segment counts as they are defined in the EDI repository. Valid values are: <ul style="list-style-type: none"> ■ yes – Repository counts are enforced. ■ no – Repository counts are not enforced. ■ multi – If the repository allows only one instance, enforce it; otherwise, treat the count as unlimited. See also "auto" on page 75.
decimal	Decimal character	Symbol used for the decimal character in the converted file. Usually a period or comma. See "Using Special Characters for Separators" on page 95 for information about how to specify values for this property.

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
decode	Where to place decoded data values	<p>Allows you to specify where to place code list value descriptions when converting EDI to XML. Valid values are:</p> <ul style="list-style-type: none"> ■ no – code list table values are not output as XML: <code><ISA15><!--I14: Interchange Usage Indicator-->P</ISA15></code> ■ comment – adds the description as a comment. For example, <code><!--Production Data--></code> in the following code: <code><ISA15><!--I14: Interchange Usage Indicator-->P<!--Production Data--></ISA15></code> ■ attribute – adds the description as an attribute: <code><ISA15 decode="Production Data"><!--I14: Interchange Usage Indicator-->P</ISA15></code> ■ text – adds the code as an attribute, and the description as element value: <code><ISA15 value="P">Production Data</ISA15></code> <p>Note that the value property must also be set to attribute to generate this output.</p>
dialect	EDI dialect	<p>Turn off this and <i>Comment element types (field)</i> to disable all comment generation. See also value .</p> <p>The EDI dialect of the file you are converting. Valid values are: EANCOM, EDIFACT, EDIG@S, HIPAA, HL7, IATA, and X12.</p> <p>This property is used only for schema generation. See “XML Schema Generation” on page 25 for more information.</p>

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
doc	Include xs:documentation	Whether or not include xs:documentation comments in the XML Schema. Valid values are yes (the default) and no. This property is used only for schema generation. See “XML Schema Generation” on page 25 for more information.
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: <ul style="list-style-type: none"> <li data-bbox="691 678 1225 843">■ yes (the default) – the value specified in the Line separator (newline) property is used to separate each segment. The normal segment output character is also generated. <li data-bbox="691 869 1225 927">■ no –linefeeds between segments are not added <li data-bbox="691 956 1253 1156">■ an 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.
element	Element separator	The character used to separate elements in a segment. See “Using Special Characters for Separators” on page 95 for information about how to specify values for this property.

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
empty	How to handle empty HL7 content	<p>Controls how XML Converters manages empty fields. Empty tokens at the first level are never written to the XML file, regardless of how this property is set.</p> <p>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 quotation marks. A missing field – that is, a field for which there is no value in the data stream – does not display quotation marks.</p> <p>Valid values for this property are:</p> <ul style="list-style-type: none"> ■ auto – The HL7 version determines how XML Converters treats empty fields: <ul style="list-style-type: none"> ■ HL7 2.2 and earlier – XML Converters behaves as if empty=empty. ■ HL7 2.3 and later – XML Converters behaves as if empty=quotes. ■ empty – All empty fields, with or without quotation marks, are treated as present but empty (use for HL7 2.2 and earlier). ■ quotes – If the field has quotes, it is treated as empty, that is, the data stream has a null value ('"'" is recognized as a marker for an empty field, for example); otherwise, it is treated as missing, that is, there is no value in the data stream (use for HL7 2.3 and later).

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
empty (cont'd)	Encoding	<p data-bbox="691 314 1253 371">Consider the following examples for different conversion scenarios:</p> <ul style="list-style-type: none"> <li data-bbox="691 395 1253 597">■ HL7 to XML, empty=empty <ul style="list-style-type: none"> <li data-bbox="729 430 1253 460">- Empty top-level elements are not output <li data-bbox="729 465 1253 527">- Empty lower-level elements are output as as empty XML elements <li data-bbox="729 534 1253 597">- Elements containing paired double-quotes are passed through unchanged <li data-bbox="691 621 1253 822">■ HL7 to XML, empty=quotes <ul style="list-style-type: none"> <li data-bbox="729 656 1253 685">- Empty top-level elements are not output <li data-bbox="729 690 1253 753">- Empty lower-level elements are not output <li data-bbox="729 760 1253 822">- Elements containing paired double-quotes are output as empty XML elements <li data-bbox="691 847 1253 1013">■ XML to HL7, empty = empty <ul style="list-style-type: none"> <li data-bbox="729 881 1253 944">- Empty elements are passed through as empty <li data-bbox="729 951 1253 1013">- Elements containing paired double-quotes are passed through unchanged <li data-bbox="691 1038 1253 1211">■ XML to HL7, empty = quotes <ul style="list-style-type: none"> <li data-bbox="729 1072 1253 1135">- Empty elements are passed through as paired double-quotes <li data-bbox="729 1142 1253 1211">- Elements containing paired double-quotes are passed through unchanged
encoding	Encoding	<p data-bbox="691 1229 1253 1317">The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML. The default is utf-8.</p>

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
field	Comment element types	<p>Creates a comment at the start of each element that includes the element’s name and number. For example, <code><!--I14: Interchange Usage Indicator--></code> in the following code:</p> <pre data-bbox="719 487 1243 539"><ISA15><!--I14: Interchange Usage Indicator-->P<!--Production Data--></ISA15></pre> <p>Turn off this and <i>Comment code list</i> (decode) to disable all comment generation.</p>
group	Use message groups if provided	<p>Adds an extra <code><GROUP></GROUP></code> around a message group in the output XML to make message groupings easier to handle with XPath for some types of documents. If you use multiple message groups within a single EDI document, turning this on may make selecting the messages within a specific group easier. This property can also be used when generating XML Schema.</p>
hexpand	Expand HL7 hex escapes	<p>In HL7 data streams, <code>\X</code> is an escape sequence used to include hex data in the stream. You can use this property (<code>hexpand=yes</code>) to expand the hex data. If the data is binary, an exception is thrown. Valid values are <code>yes</code> and <code>no</code>.</p>
hipaa	Enable HIPAA Auto-detection	<p>Whether or not XML Converters 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. Default is <code>no</code> – even if the file is recognized as a HIPAA document, it is processed as an X12 document. Can be used with atis .</p>
ignore	Ignore specific errors	<p>Allows you to specify which, if any, errors you want to ignore during XML conversion. The syntax for this field is <code>ignore=n1,n2,n3....</code> For example, <code>ignore=3,4,47</code> ignores errors 3, 4, and 47.</p>

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
indent	Whether to indent XML output	<p>Controls whether or not the XML output will be indented. Valid values are yes, no, and blank (unspecified).</p> <p>Note: If this value is unspecified, XML output is indented unless decode and field are both set to no.</p>
inter	Interactive messages	<p>Certain EDI messages have alternate batch and interactive forms, depending upon whether they are used between systems that have real-time connections. The <code>inter=yes</code> setting causes the interactive form to be 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.</p> <p>This property is used only for schema generation. See “XML Schema Generation” on page 25 for more information.</p>
invalid	Invalid character replacement	<p>Used with REPLACE value for the <code>chr</code> property to specify the character that should be used to replace invalid characters. The default (if invalid is not specified) is an underscore (“_”). Valid values are:</p> <ul style="list-style-type: none"> ■ <code>\u####</code> – To specify a Unicode value, substituting the <code>####</code> for the appropriate value. ■ <code>\d####</code> – To specify a decimal value, substituting the <code>####</code> for the appropriate value. <p>See “Using Special Characters for Separators” on page 95 for more information about how to specify values for this property.</p>

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
ldate	How to handle 'L' HL7 date	<p>Controls how XML Converters manages the <i>L</i> value (which traditionally means "local system" in HL7) if it is passed as either a date, time, datetime, or timestamp. Valid values are:</p> <ul style="list-style-type: none"> ■ header – The <i>L</i> value is replaced with the value of the MSH-7 element from the header. ■ current – The <i>L</i> value is replaced with the date and/or time that the message processing started. ■ error – The <i>L</i> value is treated as a syntax error. ■ pass – The <i>L</i> value is passed through unchanged.
leading	Ignore leading zeros on numbers	<p>Whether or not you want the XML Converter engine to ignore leading zeros on numbers. Setting ignore=yes means that leading zeros on the value in the EDI and the value in the codelist are compared without any leading zeros. So with ignore=no, "012" does not match "12"; but with ignore=yes these values do match. This applies only to codelist validation, not for handling of numbers.</p>
len	Strict validation on value lengths	<p>Checks each value against the upper and lower length limits defined in the EDI specification.</p>
long	Use long element names	<p>Whether you want to use long or short element names in your XML conversions – FTX03-TextReference (long=yes) or FTX03 (long=no), for example.</p>

Table 4-8. Properties for All EDI XML Converters

Name in URL	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>. This property is set to no (loop-prefix=no) by default.
message	Message type	The type of EDI message being converted. Valid values vary based on dialect and version. This property is used only for schema generation. See “XML Schema Generation” on page 25 for more information.
newline	Line separator	Used when converting EDI to XML, and XML to EDI when the <i>Add linefeeds between segments on write</i> property (eol) is set to yes. The default is crlf. See “Line Separator Values” on page 66 for a list of values.
opt	Treat all segments and elements as optional	If set to yes (no is the default), Stylus Studio assumes that all segments, elements, and composites are optional. Enabling this property can be useful if your provider declines to provide segments and elements that are considered mandatory according to the EDI specification, but you are aware of what the missing values are.
prefix	Namespace prefix	Namespace prefix to be added, with the Namespace URI, to the root element. The prefix alone is added to all elements.

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
release	Release (escape) symbol	The release, or escape, character. It turns off special processing of the next character. Suppose your message uses within a text description the same character that was used to separate elements. This is the character that would be used to tell the EDI processor to treat that character as a normal character and not as the end of the text. See "Using Special Characters for Separators" on page 95 for information about how to specify values for this property.
repeat	Repeat symbol	The repeat symbol for EDI dialects that use it. See "Using Special Characters for Separators" on page 95 for information about how to specify values for this property.

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
rtrim	Trim trailing delimiters	<p>Typically, most trailing delimiters are removed automatically. But in the conversion process, new trailing delimiters are sometimes created. The rtrim property controls how XML Converters manages these trailing delimiters. Valid values are:</p> <ul style="list-style-type: none"> ■ no – Trailing delimiters are not trimmed. ■ yes – Trailing delimiters are trimmed as long as performance is not affected. ■ always – Trailing delimiters are trimmed regardless of the impact on performance. <p>You can also use this property to <i>add</i> additional spaces, or padding:</p> <ul style="list-style-type: none"> ■ pad1 – Add spaces at the top-most level only. ■ pad2 – Add spaces to the first two levels only – elements and composite elements that don't contain other composites. ■ pad3 – Add spaces for every level of element. This value can create many empty elements (when converting to XML) and many empty delimiters (when converting to EDI). <p>See also “clean” on page 78.</p>
seg	Strict segment-ordering checking	<p>Whether or not you want to check segment ordering as defined by the message. If this property is set to no, message and group definitions are ignored, resulting in the possibility that data is grouped incorrectly (<GROUP_#> tags are never emitted, for example). Valid values are yes and no; the default is yes.</p>

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
segment	Segment separator	The character you want to use for segment separators. See “Using Special Characters for Separators” on page 95 for information about how to specify values for this property.
setup	Write setup segment if appropriate	Used to indicate whether or not to write the file’s setup segment when writing EDI. Used only if the EDI dialect supports an optional setup segment (the EDIFACT or IATA UNA segment, for example); otherwise, it is ignored. Valid values are yes and no; the default is yes.
strict	Strict validation mode	If strict=yes, checks that all mandatory elements are present, and ensures that no composite elements are in places where only simple elements are allowed. Also checks for extra elements at the end of segments that are not part of the specification. The default is strict=no.
strip	Strip C-style comments	Determines whether content in the incoming EDI stream wrapped in C-style /* and */ comment delimiters is ignored. Default setting is off (strip=no) since it can potentially conflict with real EDI content. HL7 files used with or generated by certain systems might include this markup, for example.
tbl	Force error if value not in code list	Generates an error if the value for an element is not in the codelist associated with that element. If this property is off (no), values are not checked for the presence of a codelist.
tertiary	Subcomponent (tertiary) separator	The character you want to use for subcomponent separators. See “Using Special Characters for Separators” on page 95 for information about how to specify values for this property.

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
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.
uri	Namespace URI	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.
user	Extension map file	The URL of the SEF file containing custom message type definitions. This property can also be used when generating XML Schema.

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
val	Enable validation	<p>Validates the EDI file (input or output) against the relevant EDI repository to ensure that all of the appropriate segments for that message are present and are in the correct order.</p> <p>When validation is enabled (val=yes), an error is generated if the EDI dialect (EDIFACT, X12, for example) isn't recognized; an error is also generated if the dialect is recognized, but the message type isn't. Missing mandatory segments, or segments not specified for a particular group will also generate errors.</p> <p>If validation is disabled (val=no) errors are not generated in the following instances, for example:</p> <ul style="list-style-type: none"> ■ If the file has an unsupported version number ■ If the message does not exist in the EDI repository; the file is still processed, but segment order checking is not performed. ■ If a segment does not exist in the EDI repository; the file is still processed, but codelist validation, and element and composite number verification is not performed. <p>Validation is enabled by default.</p>

Table 4-8. Properties for All EDI XML Converters

Name in URL	Property Name	Description
value	Where to place coded data values	<p>Allows you to specify where you want to place coded data values in XML output. Valid values are:</p> <ul style="list-style-type: none"> ■ text – outputs the coded data value (here, 00) in the text node: <code><BGN01><!--353: Transaction Set Purpose Code-->00</BGN01></code> ■ attribute – outputs the coded data value as an attribute: <code><BGN01 value="00"><!--353: Transaction Set Purpose Code--></BGN01></code>
version	Dialect version	<p>See also decode .</p> <p>The version of the dialect specified by the dialect property. Valid values vary based on dialect.</p> <p>This property is used only for schema generation. See “XML Schema Generation” on page 25 for more information.</p>

FINNISH Character Set Overrides

As described elsewhere, you can use the using the [Character repertoire override](#) property (`chr`) 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 4-9. 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 (`chr`), you can instruct the 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 may use the encodings described in the following table:

Table 4-10. 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 or IATX	UN/ECE level X (Code extension technique as defined by ISO 2022 utilizing the escape techniques in accordance with ISO 2375)
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 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.

XML Converters does no 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 – you cannot specify that the colon (:) is the element separator character by entering `converter:EDI:element=:auto=yes`, for example. Instead, you must escape special characters using the appropriate decimal or hex value. To specify a colon as a element separator character, you would use `converter:EDI:element=\u3A:auto=yes`, for example. Note that not all EDI dialects use all special characters.

See [Table 4-11, “Common Separator Characters,” on page 96](#) for a complete list of separator characters and their decimal and hex values.

Which Properties Specify Separators?

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

- [“segment”](#) on page 89
- [“element”](#) on page 80
- [“component”](#) on page 78
- [“release”](#) on page 87
- [“decimal”](#) on page 78
- [“tertiary”](#) on page 89
- [“repeat”](#) on page 87.

Restrictions for 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 you choose to set.

Commonly Used Separator Characters

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

Table 4-11. Common Separator Characters

Character	Decimal	Hex
~	\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

Table 4-11. Common Separator Characters

Character	Decimal	Hex
=	\d61	\u003D
[\d91	\u005B
]	\d93	\u005D
}	\d123	\u007B
{	\d125	\u007D
\	\d92	\u005C
	\d124	\u007C
'	\d39	\u0027
;	\d59	\u003B
"	\d34	\u0022
:	\d58	\u003A
/	\d47	\u002F
.	\d46	\u002E
,	\d44	\u002C
?	\d63	\u003F
>	\d62	\u003E
<	\d60	\u003C

Control Characters

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

Table 4-12. Control Characters

Character	Decimal	Hex	Other
NUL	\d0	\u0000	
SOH	\d1	\u0001	
STX	\d2	\u0002	

Table 4-12. Control Characters

Character	Decimal	Hex	Other
ETX	\d3	\u0003	
EOT	\d4	\u0004	
ENQ	\d5	\u0005	
ACK	\d6	\u0006	
BEL	\d7	\u0007	
BELL	\d7	\u0007	
BS	\d8	\u0008	
HT	\d9	\u0009	\t
TAB	\d9	\u0009	\t
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	\d20	\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	

Table 4-12. Control Characters

Character	Decimal	Hex	Other
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	
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	

Table 4-12. Control Characters

Character	Decimal	Hex	Other
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 4-13. Properties for EDI Processing Instructions

Processing Instruction	Name in URI	Description	Default
edi_component	component	Component value separator.	:
edi_decimal	decimal	Decimal character.	,
edi_element	element	Element separator.	+
edi_invalid	invalid		
edi_release	release	Release (escape) separator.	?
edi_repeat	repeat	Repeat symbol separator.	~
edi_segment	segment	Segment separator.	'
edi_tertiary	tertiary	Subcomponent (tertiary) separator.	&

Leave these values blank to assume the default values. XML Converters will generate 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 actual start of the file might end up looking much 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>
  ...

```

HTML Converter Properties

You can use the HTML XML Converter to convert HTML to XHTML.

XML Converter Name in URL

HTML

Table 4-14. Properties for the HTML XML Converter

<i>Name in URL</i>	<i>Property Name</i>	<i>Description</i>
encoding	Encoding	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML. The default is utf-8.
errors	Abort on errors found	Whether or not you want the adapter to fail when it encounters problems with the HTML-to-XHTML mapping. If <code>errors=no</code> , the adapter continues with the conversion, making a best guess. <code>errors=yes</code> is the default.
newline	Line separator	Used when converting HTML to XHTML. The default is <code>crlf</code> . See “Line Separator Values” on page 66 for a list of values.
warnings	Abort on warnings found	Whether or not you want the adapter to fail when it encounters potential problems with the HTML-to-XHTML mapping. <code>warnings=no</code> is the default.

Java .properties File XML Converter Properties

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

XML Converter Name in URL

JavaProps

Table 4-15. Properties for JavaProps XML Converters

<i>Name in URL</i>	<i>Property Name</i>	<i>Description</i>
encoding	Encoding	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML. The default is ISO-8859-1.
newline	Line separator	Used when converting a file to XML, and vice versa. The default is crlf. See “Line Separator Values” on page 66 for a list of values.

JSON XML Converter Properties

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

XML Converter Name in URL

JSON

Table 4-16. Properties for JSON XML Converters

<i>Name in URL</i>	<i>Property Name</i>	<i>Description</i>
indent	Indent	The level of indent you want to use for the converted XML. The default is 4.
newline	Line separator	The character that indicates the start of a new line in the document to be converted. The default is crlf. See “Line Separator Values” on page 66 for a list of values.

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.

XML Converter Name in URL

DotD

Table 4-17. Properties for DotD XML Converters

<i>Name in URL</i>	<i>Property Name</i>	<i>Description</i>
encoding	Encoding	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML. The default is utf-8.
newline	Line separator	Used when converting a file to XML, and vice versa. The default is crlf. See “Line Separator Values” on page 66 for a list of values.

Pyx Format XML Converter Properties

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

XML Converter Name in URL

Pyx

Table 4-18. Properties for Pyx XML Converters

<i>Name in URL</i>	<i>Property Name</i>	<i>Description</i>
encoding	Encoding	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML. The default is utf-8.
newline	Line separator	Used when converting a file to XML, and vice versa. The default is crlf. See “Line Separator Values” on page 66 for a list of values.

Rich Text Format XML Converter Properties

XML Converter Name in URL

RTF

Table 4-19. Properties for RTF XML Converters

<i>Name in URL</i>	<i>Property Name</i>	<i>Description</i>
encoding	Encoding	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML. The default is cp850.
newline	Line separator	Used when converting a file to XML, and vice versa. The default is crlf. See “Line Separator Values” on page 66 for a list of values.

SDI XML Converter Properties

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

XML Converter Name in URL

SDI (Super Data Interchange Format)

Table 4-20. Properties for SDI XML Converters

<i>Name in URL</i>	<i>Property Name</i>	<i>Description</i>
encoding	Encoding	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML. The default is cp850.
newline	Line separator	Used when converting SDI files to XML, and vice versa. The default is crlf. See “Line Separator Values” on page 66 for a list of values.

SYLK XML Converter Properties

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

XML Converter Name in URL

SYLK (Symbolic Link Format)

Table 4-21. Properties for SYLK XML Converters

<i>Name in URL</i>	<i>Property Name</i>	<i>Description</i>
encoding	Encoding	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML. The default is cp850.
newline	Line separator	Used when converting SYLK files to XML, and vice versa. The default is crlf. See "Line Separator Values" on page 66 for a list of values.

Tab-Separated Values XML Converter Properties

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

XML Converter Name in URL

TAB (tab-separated values)

Table 4-22. Properties for Tab-Separated Values XML Converters

<i>Name in URL</i>	<i>Property Name</i>	<i>Description</i>
collapse	Collapse consecutive separators	Whether or not you want to collapse consecutive separators – that is, separators that do not contain any data. Default is no.
double	Doubling embedded quote escapes it	Whether or not doubling an embedded quotation mark has the effect of escaping the quoted string. Default is no.
encoding	Encoding	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML.
escape	Escape character	This 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 one and including explicitly named columns in the count. If number=yes, extra columns (those after the first) are named just column.

Table 4-22. Properties for Tab-Separated Values XML Converters

Name in URL	Property Name	Description
newline	Line separator	See “Line Separator Values” on page 66 for a list of values.
number	Number rows and columns	If number=yes (no is the default), each row will also have an attribute, named row, which will contain the row number from the source document, starting from one. Also, each column, even those explicitly named, will have a column attribute numbering the column from one. Any empty columns are omitted from the output, but the numbering of subsequent columns will reflect that a column(s) was skipped.
quotes	Quote characters	A list of characters the converter should interpret as quotation characters. Double and single quote marks (" ') are the default values.
root	Root element name	The value you want to use for the root element name. Default is table.
row	Row element name	The value you want to use for the row element name. Default is row.
sep	Separator	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.

XML Converter Name in URL

Line

Table 4-23. Properties for the Whole-line Text XML Converter

<i>Name in URL</i>	<i>Property Name</i>	<i>Description</i>
encoding	Encoding	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML. The default is utf-8.
line	Line element name	Value used for the line element name. Default is line.
newline	Line separator	Used when converting a whole-line text file to XML, and vice versa. The default is crlf. See “Line Separator Values” on page 66 for a list of values.
root	Root element name	Value used for the root element name. 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.

XML Converter Name in URL

WinIni

Table 4-24. Properties for WinIni XML Converters

<i>Name in URL</i>	<i>Property Name</i>	<i>Description</i>
encoding	Encoding	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML. The default is cp1252.
newline	Line separator	Used when converting a file to XML, and vice versa. The default is crlf. See “Line Separator Values” on page 66 for a list of values.

Windows Write XML Converter Properties

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

XML Converter Name in URL

WinWrite

Table 4-25. Properties for WinWrite XML Converters

<i>Name in URL</i>	<i>Property Name</i>	<i>Description</i>
encoding	Encoding	The encoding for the input file when it is not XML; or the encoding for the output file when it is not XML. The default is utf-8.
newline	Line separator	Used when converting a file to XML, and vice versa. The default is crlf. See “Line Separator Values” on page 66 for a list of values.

Index

A

- accessing data using Stylus Studio URL schemes 18
- API
 - examples 43
 - examples of the DataDirect XML Converters Java API 43

B

- Base-64 XML Converter properties 67
- binary XML Converter properties 68

C

- command line usage 19
- contacting Technical Support 12
- control characters
 - for EDI XML Converters 97
- conventions, typographical 11
- conversion results
 - pulling 61
 - pushing 61
- converter URI scheme
 - building a converter URI 39
 - description 37
 - displayed in Stylus Studio 40
 - parts of 37
 - syntax of 37
 - using with user-defined .conv files 41
- CSV XML Converter properties 69

- custom XML conversions
 - using with converter URIs 41
- customizing file conversions 17

D

- data
 - accessing data using Stylus Studio URL schemes 18
- dBase XML Converter properties 71
- demo.cs example 43
- demo.java demonstration file 43
- DIF XML Converter properties 73
- DotD XML Converter properties 105

E

- EANCOM files
 - XML Converter properties for 74
- EDI XML Converters
 - exception handling for 24
 - proprietary EDI formats and 21
 - SEF support 21
- EDIFACT files
 - XML Converter properties for 74
- error handling
 - example 54
 - overview 23
- examples 43
 - converting CSV to XML 46
 - converting X12 to XML 52
 - converting XML to CSV 47
 - converting XML to X12 52
 - converting XSLT output to CSV 51

- creating an XML Schema from a CSV file 57
- creating an XML Schema from EDI 58
- demo.cs 43
- error handling 54
- streaming XML 49
- using a document URI resolver 59
- using custom XML conversions 48
- using SEF to convert EDI 53
- using the XML Converters Java API 43
- exception handling 24

F

- files
 - customizing file conversions 17
 - formats supported by XML Converters 15
 - generating XML Schema from 32

G

- generating XML Schema
 - example 26, 35
 - file type summary 32
 - instance documents 28
 - overview 25
 - URI parameters 29

H

- HL7 files
 - XML Converter properties for 74
- HTML XML Converter properties 102

I

- IATA files
 - XML Converter properties for 74
- instance documents
 - XML Schema generation and 28

J

- Java API
 - examples of 43
- JavaProps XML Converter properties 103
- JSON XML Converter properties 104

L

- Line XML Converter properties 112

O

- OpenEdge DotD XML Converter properties 105

P

- processing instructions
 - for EDI XML Converters 100
- pulling conversion results 61
- pushing conversion results 61
- Pyx XML Converter properties 106

R

RTF XML Converter properties 107

S

SDI XML Converter properties 108

SEF

- support in EDI XML Converters 21

separator characters

- for EDI XML Converters 95

special characters

- for EDI XML Converters 95

Standard Exchange Format. See SEF

Stylus Studio

- building a converter URI using 39

SupportLink 12

SYLK XML Converter properties 109

T

TAB XML Converter properties 110

Technical Support, contacting 12

U

URI parameters

- XML Schema generation and 29

URI schemes

- descriptions of 19

URL schemes

- the `converter` URL scheme 18

W

WinIni XML Converter properties 113

WinWrite XML Converter properties 114

X

X12 files

- XML Converter properties for 74

XML Converters

- Base-64 XML Converter properties 67

- binary XML Converter properties 68

- building a converter URI using Stylus Studio 39

- command line usage 19

- control characters for EDI XML Converters 97

- CSV XML Converter properties 69

- customizing 17

- dBase XML Converter properties 71

- descriptions of 15

- DIF XML Converter properties 73

- DotD XML Converter properties 105

- EANCOM file converter properties 74

- EDIFACT file converter properties 74

- error handling 23

- examples 43

- exception handling 24

- file formats supported by 15

- generating XML Schema with 25, 26

- HL7 file converter properties 74

- HTML XML Converter properties 102

- IATA file converter properties 74

- Java API examples 43

- JavaProps XML Converter properties 103

- JSON XML Converter properties 104

- line separator values used in 66

- Line XML Converter properties 112

- OpenEdge DotD XML Converter properties 105

- overview 15

- processing instructions for EDI XML
 - Converters 100
 - Pyx XML Converter properties 106
 - RTF XML Converter properties 107
 - SDI XML Converter properties 108
 - SEF support 21
- separator characters for EDI XML
 - Converters 95
- special characters for EDI XML Converters 95
- SYLK XML Converter properties 109
- TAB XML Converter properties 110
- WinIni XML Converter properties 113
- WinWrite XML Converter properties 114
- X12 file converter properties 74
- XML Schema
 - example of generating 35
 - generating
 - example 26
 - instance documents 28
 - overview 25
 - URI parameters 29
 - generation
 - file type summary 32
- XML Schema generation
 - example 35
 - instance documents and 28
 - URI parameters and 29