

XML PRODUCTIVITY THROUGH INNOVATION

STYLUS STUDIO 2007

USER GUIDE



© 2007 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.

Acknowledgments for Stylus Studio

Stylus Studio includes:

Xerces++ developed by the Apache Software Foundation (<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/>). Copyright © 1999-2006 the Apache Software Foundation. All rights reserved.

FOP developed by the Apache Software Foundation (<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/>). 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>. 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 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 JSON.org. Copyright (c) 2002 JSON.org. All rights reserved.

April 2007

Contents

Preface	xlvi
About This Manual	xlvi
Conventions in This Manual	xlix
Typographical Conventions	xlix
Syntax Notation	1
Information Alerts	1
Edition Alerts	1
Video Alerts	li
Video Descriptions	li
Available Documentation	li
Technical Support	lii
Chapter 1: Getting Started with Stylus Studio® 2007	1
Stylus Studio Editions	3
Stylus Studio XML Enterprise Suite	3
Stylus Studio XML Professional Suite	3
Stylus Studio Home Edition	4
Edition Alerts	4
More Information	4
Starting Stylus Studio	5
Getting Updates	6
Getting Help	6
Updating an XML Document – Getting Started	6
Opening a Sample XML Document	7
Alternatives	8
For more information	8

Contents

Updating the Text of a Sample Document	8
Displaying Line Numbers	9
Adding Elements in the Text View of a Sample Document	9
Copying and Pasting in the Text View of a Sample Document.	10
Undoing Operations in the Text View of a Sample Document	11
Inserting Indents in the Text View of a Sample Document	11
Querying in the Text View of a Sample Document.	12
Deleting and Saving Queries	14
Updating the Schema of a Sample Document	15
Creating a Sample Schema.	15
Defining a Sample Element	17
Adding an Element Reference to a Sample Schema	19
Defining an Entity in a Sample Schema	20
Exploring Other Features in a Sample Schema	20
Updating the Tree Representation of a Sample Document	21
Adding an Element to a Sample Document Tree.	22
Changing an Element's Data in a Sample Document Tree	22
Adding Attributes and Other Node Types to a Sample Document Tree	23
Adding an Entity Reference to a Sample Document Tree	24
Updating a Sample Document Using the Grid Tab	24
Modifying Values	26
Moving Around the Grid	26
Working with Stylesheets – Getting Started	27
Opening a Sample Stylesheet	27
XSLT Stylesheet Editor Quick Tour	28
Parts of the XSLT Editor	29
Exploring the XSLT Source Tab	29
Exploring the Params/Other Tab	32
XSLT Scenarios	32
Working with Scenarios.	35
About Preview	35
Working with a Sample Result Document.	35
Making a Static Web Page Dynamic by Editing XSLT	38
Importing a Sample HTML File	39
Creating the video Template	41
Instantiating the video Template	43
Making Titles Dynamic	44
Making Images Dynamic	45
Making Summaries Dynamic.	46

Using the XSLT Mapper – Getting Started	47
Opening the XSLT Mapper	47
Mapping Nodes in Sample Files	50
Saving the Stylesheet and Previewing the Result	53
Deleting Links in Sample Files	54
Defining Additional Processing in Sample Files	54
Debugging Stylesheets – Getting Started	55
Setting Up Stylus Studio to Debug Sample Files	55
Inserting a Breakpoint in the Sample Stylesheet	56
Gathering Debug Information About the Sample Files	58
The Variables Window	59
The Call Stack Window	59
The Watch Window	60
Ending Processing During a Debug Session	61
Defining a DTD – Getting Started	63
Process Overview	63
Creating a Sample DTD	63
Defining Data Elements in a Sample DTD	64
Defining the Container Element in a Sample DTD	65
Defining Structure Rules in a Sample DTD	65
Examining the Tree of a Sample DTD	67
Defining an XML Schema Using the Diagram Tab – Getting Started	68
Introduction to the Diagram Tab	69
Diagram Pane	70
Text Pane	75
Definition Browser	77
Editing Tools of the Diagram Tab	78
Menus and Tool Bars	78
In-place Editing	79
Drag-and-Drop	79
QuickEdit	80
Refactoring	81
Description of Sample XML Schema	83
Defining a complexType in a Sample XML Schema in the Diagram View	83
Defining the Name of a Sample complexType in the Diagram View	84
Adding an Attribute to a Sample complexType in the Diagram View	85
Adding Elements to a Sample complexType in the Diagram View	86
Adding Optional Elements to a Sample complexType in the Diagram View	87

Contents

Adding an Element That Contains Subelements to a complexType in the Diagram View	87
Choosing the Element to Include in a Sample complexType in the Diagram View	89
Defining Elements of the Sample complexType in the Diagram View	91
Opening Files in Stylus Studio	93
Types of Files Recognized by Stylus Studio	93
Opening Unknown File Types	94
Opening Files Stored on Third-Party File Systems	94
Modifications to Open Files	95
Using the File Explorer	95
How to Use the File Explorer to Open Files	96
Other Features of the File Explorer	96
Working with the File Explorer Filter	97
Dragging and Dropping Files in the Stylus Studio	98
Other Ways to Open Files in Stylus Studio	99
Adding File Types to Stylus Studio	100
Deleting File Types	101
Working with Projects	101
Displaying the Project Window	102
Displaying Path Names	103
Other Documents	103
Creating Projects and Subprojects	104
Saving Projects	104
Opening Projects	104
Recently Opened Projects	105
Adding Files to Projects	105
Other Ways to Add Files to Projects	106
Copying Projects	106
Rearranging the Files in a Project	107
Removing Files from Projects	107
Closing and Deleting Projects	107
Closing	107
Deleting	108
Setting a Project Classpath	108
Specifying Multiple Classpaths	108
How to Set a Project Classpath	108
Using Stylus Studio with Source Control Applications	110
Tested Source Control Applications	111
Prerequisites	111

Recursive Selection	111
Using Stylus Studio with Microsoft Visual SourceSafe	112
Using Stylus Studio with ClearCase	114
Using Stylus Studio with Zeus CVS	117
Specifying Advanced Source Control Properties	118
Customizing Tool Bars	119
Tool Bar Groups	119
Showing/Hiding Tool Bar Groups	120
Changing Tool Bar Appearance	121
Specifying Stylus Studio Options	121
Setting Module Options	121
XML Diff	122
XML Editor	122
XSLT Editor	122
Java	123
Defining Custom Tools	123
Defining Keyboard Shortcuts	125
How to Define a Keyboard Shortcut	125
Deleting a Keyboard Shortcut	126
Using Stylus Studio from the Command Line	127
Invoking Stylus Studio from the Command Line	127
Applying a Stylesheet from the Command Line	128
Executing an XQuery from the Command Line	129
Validating XML from the Command Line	130
Managing Stylus Studio Performance	130
Troubleshooting Performance	131
Changing the Schema Refresh Interval	131
Checking for Modified Files	132
Changing the Recursion Level or Allocated Stack Size	133
Automatically Opening the Last Open Files	133
Configuring Java Components	134
Stylus Studio Modules That Require Java	134
Settings for Java Debugging	135
Verifying the Current Java Virtual Machine	135
Downloading Java Components	135
Modifying Java Component Settings	136
How Auto Detect Works	136
About JVM Parameters	137
About JDK Parameters	137

How to Modify Java Component Properties	137
Chapter 2: Editing and Querying XML	139
Creating XML Documents	140
Using the XML Editor	140
Other Ways to Create XML	140
Using Document Wizards to Create XML	141
How to Use a Document Wizard	141
Creating XML from XML Schema	141
Creating XML from DTD	142
Creating XML from HTML	142
Updating XML Documents	143
Choosing a View	143
For More Information	144
Saving Your Work	144
Ensuring Well-Formedness	144
Reverting to Saved Version	144
Updating Java Server Pages as XML Documents	145
Using the Text Editor	145
Text Editing Features	146
Simple Text Editing	146
Code Folding	146
Sense:X	147
Indent	148
Line Wrap	148
Spell Checking	149
Font	149
Comments	149
Bookmarks	149
Search/Replace	150
Use of Colors in the Text Tab	151
How to Change Colors	151
Using the Spell Checker	152
Default Spell Checking	152
Manual Spell Checking	152
Specifying Spell Checker Settings	153
How to Spell Check a Document	154
Using the Personal Dictionary	155

Moving Around in XML Documents	156
Line Numbers	156
Bookmarks	157
Tags	157
Find	157
Updating DOM Tree Structures	158
Displaying All Nodes in the Tree View	159
Adding a Node in the Tree View	159
Deleting a Node in the Tree View	160
Moving a Node in the Tree View	160
Changing the Name or Value of a Node in the Tree View	160
Obtaining the XPath for a Node	161
Using the Grid Tab	162
Layout of the Grid Tab	163
Features of the Grid Tab	163
Expanding and Collapsing Nodes	164
Collapsing Empty Nodes	164
Renaming Nodes	166
Resizing Columns	166
Showing Row Tag Names	167
Moving Around the Grid Tab	167
Selecting Items in the Grid	168
How Grid Changes Affect the XML Document	168
Types of Changes that Affect the Document	169
Working with Rows	169
Reordering Rows	170
Adding and Deleting Rows	170
Working with Columns	171
Selecting a Column	171
Adding Columns	171
Deleting Columns	172
Reordering Columns	172
Renaming Columns	173
Changing a Value	173
Working with Tables	173
Adding a Nested Table	174
Moving a Nested Table	175
Deleting a Table	175
Sorting a Table	176

Contents

Copying a Table as Tab-Delimited Text	176
Diffing Folders and XML Documents	177
Overview.	178
Sources and Targets	179
The Diff Configuration File	179
What Diffs Are Calculated?	179
Tuning the Diffing Algorithm	180
When Does the Diff Run?	181
Running the Diff Manually	182
Symbols and Background Colors.	182
Diffing Folders	183
Features	184
How to Diff Folders	185
How to Diff Documents from the Diff Folders Dialog Box	187
The XML Diff Viewer	187
Split View - Tree	188
Split View - Text	189
Merged View	190
View Symbols and Colors	191
The XML Diff Viewer Tool Bar	191
Tools for Working with Documents	194
Removing a Target Document	194
Diffing a Pair of XML Documents	194
How to Diff a Pair of Documents	195
Diffing Multiple Documents.	196
Document Focus.	196
Symbols Used in the Target Document Window.	197
How to Diff Multiple Documents	200
Modifying Default Diff Settings.	201
Opening the Options Dialog Box.	202
Engine Settings.	203
Presentation Options	205
Running the Diff Tool from the Command Line	205
Restrictions.	206
Usage	206
Using Schemas with XML Documents.	208
Associating an External Schema With a Document.	208
Having Stylus Studio Generate a Schema	209
Validating XML Documents	209

Updating a Document’s Schema	210
Removing the Association Between a Document and a Schema	210
Converting XML to Its Canonical Form	211
Querying XML Documents Using XPath	211
Printing XML Documents	211
Saving XML Documents	212
Options for Saving Documents	212
More About Backup Files	212
Opening a Backup File	213
Chapter 3: Converting Non-XML Files to XML	215
Introduction	216
Accessing Conversion Tools	216
Other Ways to Convert Files to XML	217
DataDirect XML Converters	217
DataDirect XML Converters in Stylus Studio	217
Types of XML Converters	218
XML Converters Can Be Configured	220
Using an XML Converter to Open a Non-XML File as XML	221
Saving an XML File in Another Format	222
Custom XML Converters	223
Creating a Custom XML Conversion Definition	224
Choosing an Input File	224
The Custom XML Conversion Definition Editor	226
Document Pane	227
Example – .txt Files	227
Display of Delimiting and Control Characters	229
Field Names	230
Document Pane Display Features	231
Moving Around the Document	233
Properties Window	234
How Properties are Organized	235
Properties for Fixed-Width and Line-Oriented Input Files	235
Schema Pane	236
Parts of an Input File	237
Regions	237
Region Types	238
Managing Regions	238
Rows	238

Contents

Fields	239
Component and Sub-Component Fields	239
Working with Regions	239
Converting the Region Type	240
How to Convert a Region Type	242
Adjusting Fixed-Width Regions	242
Example	243
Defining and Joining Regions	244
Defining a Region	244
Joining Regions	246
Controlling Region Output	246
Working with Fields	247
Naming Fields	247
Using the Element Name Source Property	248
More About Using Rows for Field Names	249
How to Name Fields	249
Defining Fields	250
Creating Notes for Fields	253
Component and Sub-Component Fields	253
Controlling XML Output	255
Specifying Element Names	255
Specifying Format	256
Omitting Regions and Fields, and Rows	256
Pattern Matching	257
Example	257
Sample Regular Expressions	258
Specifying Multiple Match Patterns	259
Working with Nodes	260
Using Lookup Lists	262
Defining Lookup Lists	263
Working with Lookup Lists	265
Using Key=Value Characters	265
Creating a Custom XML Conversion Definition	266
Specifying File Settings	266
How to Create a Custom XML Conversion Definition	267
Using Custom XML Conversion Definitions in Stylus Studio	268
How to Open a File Using a Custom XML Conversion Definition	268
Working with EDI Conversions	271
Supported EDI Dialects	271

Creating Custom EDI Message Types	272
Overview	272
Specifying the Extension File Location	273
The ex.xsd XML Schema	274
The Root Element	274
The Message Element	274
The Group Element	275
The Segment Element	275
Composite Fields	276
Elements	276
Understanding Separator Characters	277
How Stylus Studio Sets Separator Characters	277
Setting Processing Instruction	278
Syntax	278
Ways to Specify Control Characters	279
Invalid Separator Characters	283
Validating XML from/to EDI	283
The Converter URL Scheme	284
Where You Use Converter URLs	284
Specifying a Converter URL	284
Example – Converter URL with a DataDirect XML Converter	284
Example – Converter URL with a Custom XML Conversion Definition	285
Converter URL Syntax	285
XML Converter Properties	286
Where Converter URLs are Displayed in Stylus Studio	287
Using Stylus Studio to Build a Converter URL	288
Using the URL in the Select XML Converter Dialog Box	288
Using the URL in the Properties Window	289
Custom XML Conversion Definitions Properties Reference	290
Input File Properties	290
XML Output URL Properties	291
Region Type Properties	293
Row Element Name Properties	296
Field Element Name Properties	297
Data Type Properties (by data type)	299
Common Properties	299
BCD Datatype Properties	300
Binary Datatype Properties	301
Boolean Datatype Properties	301

Byte Datatype Properties	304
Comp3 Datatype Properties	305
Date Datatype Properties	305
DateTime Datatype Properties	308
Decimal Datatype Properties	308
Double Datatype Properties	309
Float Datatype Properties	310
Integer Datatype Properties	311
Long Datatype Properties	312
Number Datatype Properties	313
Short Datatype Properties	316
String Datatype Properties	317
Time Datatype Properties	318
Zoned Datatype Properties	320
Specifying Control Characters	320
Chapter 4: Working with XSLT	325
Getting Started with XSLT	325
What Is XSLT?	326
What XSLT Versions Does Stylus Studio Support?	326
What Is a Stylesheet?	327
Example of a Stylesheet	327
About Stylesheet Contents	330
What Is a Template?	330
Contents of a Template	331
Determining Which Template to Instantiate	332
How the select and match Attributes Are Different	333
How the XSLT Processor Applies a Stylesheet	333
Instantiating the First Template	334
Selecting Source Nodes to Operate On	335
Controlling the Order of Operation	336
Omitting Source Data from the Result Document	337
When More Than One Template Is a Match	338
When No Templates Match	338
Controlling the Contents of the Result Document	339
Specifying Result Formatting	339
Creating New Nodes in the Result Document	340
Controlling White Space in the Result	340
Specifying XSLT Patterns and Expressions	341

Examples of Patterns and Expressions	341
Frequently Asked Questions About XSLT	343
Sources for Additional XSLT Information	344
Benefits of Using Stylus Studio	345
Structural Data View	345
Sophisticated Editing Environment	346
XSLT and Java Debugging Features	347
Integrated XML Parser/XSLT Processor	349
Tutorial: Understanding How Templates Work	349
Creating a New Sample Stylesheet	350
Understanding How the Default Templates Work	353
Instantiating the Template That Matches the Root Node	354
Instantiating the Root/Element Default Template	355
Instantiating the Text/Attribute Default Template	356
Illustration of Template Instantiations	357
Editing the Template That Matches the Root Node	358
Creating a Template That Matches the book Element	358
Creating a Template That Matches the author Element	359
Working with Stylesheets	360
About the XSLT Editor	361
Creating Stylesheets	362
Creating a Stylesheet from HTML	362
Specifying Stylesheet Parameters and Options	363
Applying Stylesheets	366
About Applying Stylesheets	366
Results of Applying a Stylesheet	367
Applying Stylesheets to Large Data Sets	369
Creating a Scenario	369
Cloning Scenarios	371
Saving Scenario Meta-Information	371
Applying a Stylesheet to Multiple Documents	372
Applying the Same Stylesheet in Separate Operations	372
Applying a Stylesheet to Multiple Documents in One Operation	372
About Stylesheet Contents	373
Contents Provided by Stylus Studio	373
Contents You Can Add	373
Updating Stylesheets	374
Dragging and Dropping from Schema Tree into XSLT Editor	374
Using Sense:X Automatic Tag Completion	375

Contents

Using Sense:X to Ensure Well-Formed XML	375
Using Standard Editing Tools	376
Saving Stylesheets	376
Using Updated Stylesheets	377
Specifying Extension Functions in Stylesheets	377
Using an Extension Function in Stylus Studio	378
Basic Data Types	379
Declaring an XSLT Extension Function	379
Working with XPath Data Types	380
Declaring an Extension Function Namespace	380
Invoking Extension Functions	381
Finding Classes and Finding Java	381
Debugging Stylesheets That Contain Extension Functions	381
Working with Templates	382
Viewing Templates	382
Viewing a List of Templates	383
Viewing a Specific Template	383
Checking if a Template Generates Output	384
Using Stylus Studio Default Templates	384
Contents of a New Stylesheet Created by Stylus Studio	384
About the Root/Element Built-In Template	385
About the Text/Attribute Built-In Template	385
Creating Templates	386
Saving a Template	386
Applying Templates	387
Updating Templates	387
Deleting Templates	387
Using Third-Party XSLT Processors	387
How to Use a Third-Party Processor	388
Passing Parameters	390
Setting Default Options for Processors	390
Validating Result Documents	392
Post-processing Result Documents	393
Generating Formatting Objects	394
Developing Stylesheets That Generate FO	395
Troubleshooting FOP Errors	395
Viewing the FO Sample Application	396
Deploying Stylesheets That Generate FO	398
Example	398

Using Apache FOP to Generate NonPDF Output	399
Generating Scalable Vector Graphics	400
About SVG Viewers	400
Running the SVG Example	400
Generating Java Code for XSLT	401
Scenario Settings	402
Choosing Scenarios	403
Java Code Generation Settings	404
How to Generate Java Code for XSLT	404
Compiling Generated Code	406
How to Modify the Stylus Studio Classpath	406
How to Compile and Run Java Code in Stylus Studio	407
Deploying Generated Code	408
XSLT Instructions Quick Reference	408
xsl:apply-imports	410
xsl:apply-templates	410
Format	410
Description	410
Example	411
xsl:attribute	411
Format	411
Description	411
Example	412
xsl:attribute-set	412
Format	412
Description	412
Example	414
xsl:call-template	414
Format	414
Description	414
xsl:character-map	414
Format	415
Description	415
Example	416
xsl:choose	417
Format	417
Description	417
xsl:comment	418
Format	418

Contents

Description	418
Example	418
xsl:copy	418
Format	418
Description	418
Example	419
xsl:copy-of	419
Format	419
Description	419
xsl:decimal-format	420
Format	420
Description	420
xsl:element	421
Format	421
Description	421
Example	422
xsl:fallback	422
xsl:for-each	422
Format	422
Description	422
Example	423
xsl:for-each-group	424
Format	424
Description	424
xsl:function	425
Format	425
Description	425
Example	426
xsl:if	426
Format	426
Description	426
Example	427
xsl:import	427
Format	427
Description	427
xsl:import-schema	427
Format	428
Description	428
Example	429

xsl:include	429
Format	429
Description	429
xsl:key	430
Format	430
Description	430
xsl:message	431
Format	431
Description	431
xsl:namespace-alias	432
Format	432
Description	432
xsl:number	432
Format	432
Description	432
Example	433
xsl:otherwise	433
xsl:output	433
Format	433
Description	434
xsl:output-character	436
Format	436
Description	436
Example	436
xsl:param	436
Format	436
Description	436
Passing parameters to templates	437
xsl:preserve-space	438
xsl:processing-instruction	438
Format	438
Description	438
Example	438
xsl:sequence	439
Format	439
Description	439
Example	439
xsl:sort	439
Format	439

Contents

Description	440
Example	441
xsl:strip-space	441
xsl:stylesheet	442
Format	442
Description	442
xsl:template	442
Format	442
Description	442
xsl:text	444
Format	444
Description	444
Examples	444
xsl:transform	445
xsl:value-of	445
Format	445
Description	445
Example	445
xsl:variable	446
Format	446
Description	446
xsl:when	447
xsl:with-param	447
Format	447
Description	447
Example	448

Chapter 5: Creating XSLT Using the XSLT Mapper 449

Overview of the XSLT Mapper	450
Example	451
Graphical Support for Common XSLT Instructions and Expressions	452
Setting Options for the XSLT Mapper	453
Simplifying the Mapper Canvas Display	454
Other Mapper Display Features	456
Exporting Mappings	456
Searching Document Panes	457
Ensuring That Stylesheets Output Valid XML	457
Steps for Mapping XML to XML	457

Source Documents	458
Choosing Source Documents	458
Source Documents and XML Instances	459
Types of associations	459
Source document icons	461
How to change a source document association.	461
How to Add a Source Document.	462
How to Remove a Source Document	463
How Source Documents are Displayed.	463
Document structure symbols	464
Getting source document details.	464
Target Structures	465
Using an Existing Document.	465
Building a Target Structure	465
Modifying the Target Structure.	467
Adding a Node	467
Removing a Node	467
Mapping Source and Target Document Nodes	468
Preserving Mapper Layout	468
Left and Right Mouse Buttons Explained.	468
How to Map Nodes	469
Removing Source-Target Maps.	470
Working with XSLT Instructions in XSLT Mapper	470
What XSLT Instructions Are Represented Graphically	471
Instruction Block Ports	471
Specifying Values for Ports	472
Understanding Input Ports.	472
Specifying Values for Input Ports.	473
Red Input Ports	473
The Flow Port	473
Adding an Instruction Block to the XSLT Mapper.	474
Notes About Creating Instruction Blocks.	474
xsl:if and xsl:choose	475
Processing Source Nodes.	476
XPath Function Blocks	476
Parts of a Function Block	477
Types of Function Blocks	477
XPath Mathematical Functions.	478
Creating a Function Block	479

Contents

Deleting a Function Block	479
Logical Operators	479
Setting a Text Value	480
Example	480
How to Set a Text Value on the Mapper Canvas	480
How to Set a Text Value on the Target Node	481
Defining Java Functions in the XSLT Mapper	482
About Adding Java Class Files	482
Creating and Working with Templates	483
What Happens When You Create a Template	483
How to Create a Named or Matched Template	484
Creating an XSLT Scenario	485
Overview of Scenario Features	485
XML Source Documents	485
Global Parameters	486
XSLT Processors	488
Performance Metrics Reporting	488
Result Document Validation	489
Post-Processing Result Documents	489
How to Create a Scenario	489
How to Run a Scenario	490
How to Clone a Scenario	491
Chapter 6: Debugging Stylesheets	493
Steps for Debugging Stylesheets	494
Using Breakpoints	494
Inserting Breakpoints	494
Removing Breakpoints	495
Start Debugging	495
Viewing Processing Information	495
Watching Particular Variables	496
Evaluating XPath Expressions in the Current Processor Context	496
Obtaining Information About Local Variables	496
Determining the Current Context in the Source Document	497
Displaying a List of Process Suspension Points	497
Displaying XSLT Instructions for Particular Output	498
Using Bookmarks	498
Determining Which Template Generated Particular Output	499
Determining the Output Generated by a Particular Template	499

Profiling XSLT Stylesheets	500
About Metrics	501
Enabling the Profiler	501
Displaying the XSLT Profiler Report	502
Handling Parser and Processor Errors	503
Debugging Java Files	503
Requirements for Java Debugging	503
Setting Options for Debugging Java	504
Using the Java Editor	505
Stylus Studio and the JVM	506
Example of Debugging Java Files	506
Setting Up to Debug Sample Java/XSLT Application	507
Inserting a Breakpoint in the Sample Java/XSLT Application	507
Gathering Debug Information About the Sample Java/XSLT Application	508
Chapter 7: Defining XML Schemas	509
What Is an XML Schema?	510
Reference Information	510
Creating an XML Schema in Stylus Studio	510
Creating Your Own XML Schema	511
Creating XML Schema from a DTD	511
Using the DTD to XML Schema Document Wizard	511
Using the DTD to XML (Trang) Document Wizard	512
Creating XML Schema from an XML Document	516
Using the XML to XML Schema Document Wizard	516
Using the Create Schema from XML Content Feature	517
Displaying the New XML Schema	518
Creating XML Schema from EDI	519
Wizard Options	519
Running an EDI to XML Schema Document Wizard	520
Working with XML Schema in Stylus Studio	521
Views in the XML Schema Editor	523
Validating XML Schema	526
Updating XML Schema Associated with a Document	526
Viewing Sample XML	526
Using XML Schema in XQuery and XSLT Mapper	528
Printing	528
Printing XML Schema	528
Printing XML Schema Documentation	528

Contents

Node Properties	529
Working with Properties in the Diagram	529
Getting Started with XML Schema in the Tree View	530
Description of Sample XML Schema	530
Tips for Adding Nodes	531
Defining a complexType in a Sample XML Schema in the Tree View	531
Defining the Name of the Sample complexType in the Tree View	531
Adding an Attribute to a Sample complexType in the Tree View	532
Adding Elements to a Sample complexType in the Tree View	532
Adding Optional Elements to a Sample complexType in the Tree View	533
Adding an Element That Contains Subelements to a complexType in the Tree View	533
Choosing the Element to Include in the Sample complexType in the Tree View	534
Defining Elements of the Sample complexType in the Tree View	535
Defining simpleTypes in XML Schemas	536
About simpleTypes in XML Schemas	536
Examples of simpleTypes in an XML Schema	537
Defining a simpleType in the Diagram View	538
Before You Begin	538
Defining an Atomic simpleType	538
Specifying a Restriction for a simpleType – QuickEdit	539
Specifying a Restriction for a simpleType – Manually	540
Defining List and Union simpleTypes	541
Defining a simpleType in the Tree View	542
About Facet Types for simpleTypes	544
Defining List and Union simpleTypes in the Tree View	545
Defining complexTypes in XML Schemas	546
Defining complexTypes That Contain Elements and Attributes – Diagram View	547
Adding Nodes to a complexType	547
Choosing an Element	548
Including All Elements	549
Specifying the Sequence of Elements	549
Reordering Nodes	550
Combining the Sequence and Choice Modifiers	550
Defining complexTypes That Contain Elements and Attributes – Tree View	551
Defining complexTypes That Mix Data and Elements	553
Diagram View	553
Tree View	554
Defining complexTypes That Contain Only Attributes	555
Diagram View	555

Tree View	556
Defining Elements and Attributes in XML Schemas	556
Defining Elements That Carry Attributes and Contain Data in XML Schemas	557
Diagram View	557
Tree View	559
Defining Elements That Contain Subelements in XML Schemas	560
Diagram View	560
Tree View	561
Adding an Identity Constraint to an Element	561
Example of an Identity Constraint	562
Diagram View	563
Tree View	564
Defining Groups of Elements and Attributes in XML Schemas	565
Defining Groups of Elements in XML Schemas – Diagram View	565
Alternative	566
Defining Groups of Elements in XML Schemas – Tree View	566
Defining attributeGroups in XML Schemas – Diagram View	567
Defining attributeGroups in XML Schemas – Tree View	568
Adding Comments, Annotation, and Documentation Nodes to XML Schemas	569
Comments	569
Annotations	569
Diagram View	569
Tree View	570
Moving a Comment or Annotation	570
Example	571
Defining Notations	572
Diagram View	572
Tree View	572
Referencing External XML Schemas	573
Ways to Reference XML Schemas	573
Including an XML Schema	573
Importing an XML Schema	574
Redefining an XML Schema	574
Where You Can Reference XML Schemas	574
What to Do Next	575
Referencing XML Schemas in the Diagram View	575
Referencing XML Schemas in the Tree View	577
Redefining Nodes	578
Extensions and Restrictions	578

Contents

Specifying Restriction Facets	578
How to Redefine a Node	579
Generating Documentation for XML Schema	580
XS3P Stylesheet Overview	581
XS3P Stylesheet Features	582
XS3P Stylesheet Settings	583
Modifying the XS3P Stylesheet	584
Saving XML Schema Documentation	584
Printing XML Schema Documentation	584
Generating JAXB Classes	585
What Stylus Studio Generates	586
How to Generate JAXB Classes	586
Compiling JAXB Class Files	587
About XML Schema Properties	587
About xsd:schema Properties	588
Element and Element Reference Properties in XML Schemas	590
Attribute and Attribute Reference Properties in XML Schemas	592
Group Properties in XML Schemas	594
Model Group Properties in XML Schemas	594
Complex and simpleType Properties in XML Schemas	596
Restriction and Extension Type Properties in XML Schemas	597
Content Type Properties in XML Schemas	597
Aggregator Type Properties in XML Schemas	598
Facet Type Properties in XML Schemas	599
Notation Type Properties in XML Schemas	600
Include Type Properties in XML Schemas	600
Import Type Properties in XML Schemas	601
Redefine Type Properties in XML Schemas	601
Identity Constraint Type Properties in XML Schemas	601
Constraint Element Type Properties in XML Schemas	602
Documentation Type Properties in XML Schemas	602
Chapter 8: Defining Document Type Definitions	603
What Is a DTD?	604
Creating DTDs	604
About Editing DTDs	605
Restrictions	605
About Modifiers in Element Definitions in DTDs	606
Description of Element Modifiers in DTDs	606

Simple Example of Aggregating Modifiers in DTDs	607
More Complex Example of Aggregating Modifiers in DTDs	608
Aggregating Modifiers to Allow Any Order and Any Number in DTDs	608
Defining Elements in DTDs	609
Defining Elements in the DTD Tree Tab	610
Specifying That an Element Can Have an Attribute in DTDs	611
Specifying That an Element is Required in DTDs	611
Specifying That an Element is Optional in DTDs	612
Specifying That Multiple Instances of An Element Are Allowed in DTDs	613
Specifying That An Element Can Contain One of a Group of Elements in DTDs	615
Specifying That an Element Can Contain One or More Elements in DTDs	616
Specifying That an Element Can Contain Data in DTDs	618
Moving, Renaming, and Deleting Elements in DTDs	618
Defining General Entities and Parameter Entities in DTDs	618
Steps for Defining Entities in DTDs	619
General Entity Example in a DTD	620
Parameter Entity Example in a DTD	621
Inserting White Space in DTDs	621
Adding Comments to DTDs	621
About Node Properties in DTDs	622
Description of Element Properties in DTDs	623
Description of Attribute Properties in DTDs	623
Description of Entity and Parameter Entity Properties in DTDs	625
Associating an XML Document with an External DTD	626
Moving an Internal DTD to an External File	626
Chapter 9: Writing XPath Expressions	629
About the XPath Processor	630
Where You Can Use XPath Expressions	630
About XPath	630
Benefits of XPath	631
Internationalization	632
Restrictions on Queries	632
Using the XPath Query Editor	632
Parts of the XPath Query Editor	633
Displaying the XPath Query Editor	634
Customizing Syntax Coloring	635
Working with XPath Queries	635
Executing the Query	636

Contents

Creating a New Query	636
Deleting a Query	636
Working with Query Results	637
Opening Query Results as a New Document	637
Working with Namespaces	638
Viewing/Changing Namespace Prefixes	638
Sample Data for Examples and Practice	639
About XML Document Structure	639
A Sample XML Document	640
Tree Representation of a Sample XML Document	640
Steps for Trying the Sample Queries	643
Getting Started with Queries	643
Obtaining All Marked-Up Text	644
Obtaining a Portion of an XML Document	644
Obtaining All Elements of a Particular Name	645
Obtaining All Elements of a Particular Name from a Particular Branch	646
Different Results from Similar Queries	647
Queries That Return More Than You Want	647
Specifying Attributes in Queries	648
Restrictions	649
Attributes and Wildcards	649
Filtering Results of Queries	649
Quotation Marks in Filters	650
More Filter Examples	650
How the XPath Processor Evaluates a Filter	651
Multiple Filters	651
Filters and Attributes	652
Wildcards in Queries	652
Restrictions	653
Attributes	653
Calling Functions in Queries	653
Case Sensitivity and Blank Spaces in Queries	654
Precedence of Query Operators	655
Specifying the Nodes to Evaluate	656
Understanding XPath Processor Terms	657
Axis	657
Context Node	657
Context Node Set	657

Current Node	657
Document Element	657
Filter	658
Location Path Expression	658
Location Step	658
Node Test	658
Root Node	658
Starting at the Context Node	659
About Root Nodes and Document Elements	659
Starting at the Root Node	659
Descending Along Branches	660
Explicitly Specifying the Current Context	661
Specifying Children or Descendants of Parent Nodes	662
Examples of XPath Expression Results	662
Syntax for Specifying an Axis in a Query	663
Supported Axes	664
About the child Axis	664
About the descendant Axis	665
About the parent Axis	665
About the ancestor Axis	665
About the following-sibling Axis	666
About the preceding-sibling Axis	666
About the following Axis	666
About the preceding Axis	667
About the attribute Axis	667
About the namespace Axis	668
About the self Axis	668
About the descendant-or-self Axis	668
About the ancestor-or-self Axis	669
Axes That Represent the Whole XML Document	669
Handling Strings and Text	670
Searching for Strings	670
Finding Identical Strings	670
Case Sensitivity	671
Finding Strings That Contain Strings You Specify	671
Finding Substrings That Appear Before Strings You Specify	671
Finding Substrings That Appear After Strings You Specify	672
Finding Substrings by Position	672

Contents

Manipulating Strings	673
Concatenating Strings	673
Determining the Number of Characters in a String	673
Normalizing Strings	674
Replacing Characters in Strings with Characters You Specify	674
Converting Objects to Strings	675
Finding Strings That Start with a Particular String	676
Obtaining the Text Contained in a Node	676
Specifying Boolean Expressions and Functions	677
Using Boolean Expressions	677
Case Sensitivity	677
Examples	677
Calling Boolean Functions	678
Converting an Object to Boolean	678
Obtaining Boolean Values	679
Determining the Context Node Language	679
Specifying Number Operations and Functions	680
Performing Arithmetic Operations	680
Calling Number Functions	681
Converting an Object to a Number	681
Obtaining the Sum of the Values in a Node Set	682
Obtaining the Largest, Smallest, or Closest Number	682
Comparing Values	683
About Comparison Operators	684
How the XPath Processor Evaluates Comparisons	684
Comparing Node Sets	685
Two Node Sets	685
A Node Set and a Number	685
A Node Set and a String	686
A Node Set and a Boolean Value	686
Comparing Single Values With = and !=	686
Comparing Single Values With <=, <, >, and >=	687
Priority of Object Types in Comparisons	687
Examples of Comparisons	688
Operating on Boolean Values	688
Finding a Particular Node	688
About Node Positions	689
Determining the Position Number of a Node	689
Positions in Relation to Parent Nodes	690

Finding Nodes Relative to the Last Node in a Set	691
Finding Multiple Nodes.	691
Examples of Specifying Positions.	692
Finding the First Node That Meets a Condition	692
Finding an Element with a Particular ID.	692
The id() Function’s Argument	693
Unique IDs	693
Obtaining Particular Types of Nodes By Using Node Tests.	694
About the Document Object	695
Getting Nodes of a Particular Type	695
Obtaining a Union	695
Obtaining Information About a Node or a Node Set.	697
Obtaining the Name of a Node	697
Wildcards	697
Obtaining Namespace Information	697
Obtaining the Namespace URI.	698
Obtaining the Local Name	698
Obtaining the Expanded Name.	698
Specifying Wildcards with Namespaces.	699
Examples of Namespaces in Queries	699
Obtaining the URI for an Unparsed Entity	700
Determining the Number of Nodes in a Collection.	700
Determining the Context Size	700
Using XPath Expressions in Stylesheets	701
Using Variables.	701
Obtaining System Properties.	701
Determining If Functions Are Available.	702
Obtaining the Current Node for the Current XSLT Template	702
Finding an Element with a Particular Key	703
Generating Temporary IDs for Nodes.	705
Format.	705
Accessing Other Documents During Query Execution	705
Format of the document() Function.	706
When the First Argument is a Node Set	706
Specification of Second Argument	706
Example of Calling the document() Function.	707
XPath Quick Reference	707
XPath Functions Quick Reference	708

Contents

XPath Syntax Quick Reference	712
Axes	712
Node Tests	712
Filters	713
Location Steps	713
XPath Expression	713
XPath Abbreviations Quick Reference	713
Chapter 10: Working with XQuery in Stylus Studio	717
Getting Started with XQuery in Stylus Studio	718
What is XQuery?	718
Example	718
Sources for Additional XQuery Information	719
What is an XQuery?	719
The Stylus Studio XQuery Editor	719
XQuery Source Tab	719
Mapper Tab	721
XQuery Source and Mapper Tab Interaction	722
An XQuery Primer	723
What is XQuery For?	723
Your First XQueries	723
Accessing XML Documents with XQuery	724
Handling URLs	725
The videos.xml Document	725
XQuery and XPath	726
XPath Query Editor	729
Introduction to FLWOR Expressions	730
Generating XML Output with XQuery	731
Accessing Databases with XQuery	733
Understanding FLWOR Expressions	734
Simple XQuery FLWOR Expressions	734
The Principal Parts of an XQuery FLWOR Expression	735
F is for For	735
L is for Let	739
W is for Where	743
O is for Order By	744
R is for Return	745
Other Parts of the XQuery FLWOR Expression	747
Declaring XQuery Types	748

XQuery Position Variables	748
Multiple Assignments	749
Grouping	749
Building an XQuery Using the Mapper	750
Process Overview	751
Working with Existing XQueries	751
Saving the Mapping	751
Source Documents	752
Choosing Source Documents	752
Source Documents and XML Instances	752
Source document icons	754
How to Change a Source Document Association	754
How to Add a Source Document	755
How to Remove a Source Document	756
How Source Documents are Displayed	756
Document structure symbols	757
Getting source document details	757
Specifying a Target Structure	757
Using an Existing Document	758
Building a Target Structure	758
Modifying the Target Structure	760
Adding a Node	760
Removing a Node	760
Setting a Text Value	760
Mapping Source and Target Document Nodes	761
Preserving Mapper Layout	761
Left and Right Mouse Buttons Explained	762
How to Map Nodes	763
Link Lines Explained	763
Removing Source-Target Map	766
Simplifying the Mapper Canvas Display	767
Other Mapper Display Features	768
Exporting Mappings	768
Searching Document Panes	769
FLWOR Blocks	769
Parts of a FLWOR Block	770
Creating a FLWOR Block	771
Function Blocks	772
Standard Function Block Types	772

Contents

Creating a Function Block	772
Parts of a Function Block	773
User-Defined Functions	774
concat Function Blocks	775
IF Blocks	776
Condition Blocks	776
Working with the XQuery collection() Function	777
Using the collection() Function in Stylus Studio	778
How the collection() Function is Processed	778
Database Connections	778
Handling Invalid Characters	779
Creating a Database Connection	779
Choosing a Database	779
Using the Server URL Field	780
Username and Password	780
How to Create a Database Connection	780
Creating a collection() Statement	782
collection() Function Syntax	782
What Happens When You Create a collection() Statement?	782
Creating Multiple Connections	783
How to Create a collection() Statement	784
Other Ways to Register a Database Configuration	785
Choosing a Database Object	786
Debugging XQuery Documents	787
Using Breakpoints	788
Inserting Breakpoints	788
Removing Breakpoints	788
Start Debugging	788
Viewing Processing Information	789
Watching Particular Variables	789
Evaluating XPath Expressions in the Current Processor Context	789
Obtaining Information About Local Variables	790
Displaying a List of Process Suspension Points	790
Displaying XQuery Expressions for Particular Output	790
Using Bookmarks	791
Inserting	791
Removing	791
Moving Focus	791

Profiling XQuery Documents	792
About Performance Metrics	793
Enabling the Profiler	793
Displaying the XQuery Profiler Report	794
Using DataDirect XQuery® Execution Plans	795
Query Plans in Stylus Studio	795
Example of a Query Plan	795
Parts of a Query Plan	796
Navigation	797
Query Plan Toolbar	798
Formatting	798
Saving a Query Plan as HTML	798
Displaying a Query Plan	799
Prerequisites	799
How to display a query plan	799
Optimizing Your XQuery	799
Creating an XQuery Scenario	800
Specifying XML Input	800
Selecting an XQuery Processor	802
Setting Default Options for Processors	804
Setting Values for External Variables	805
Performance Metrics Reporting	806
Validating XQuery Results	806
How to Create a Scenario	808
How to Run a Scenario	809
How to Clone a Scenario	809
Generating XQuery Documentation	810
Documentation Defaults	811
Syntax and Usage	811
Save the XQuery Document	813
ActiveX Controls	813
Viewing Code Samples	814
How to Generate XQuery Documentation	814
Using XQuery to Invoke a Web Service	816
Requirements	816
Using the Saxon Processor	816
Invoking a SOAP Request in an XQuery	817
Invoking Multiple SOAP Requests	818
Rules	818

Contents

How to Invoke Multiple SOAP Requests in the Same XQuery	818
Generating Java Code for XQuery	819
Scenario Settings	820
Choosing Scenarios	821
Java Code Generation Settings	822
How to Generate Java Code for XQuery	822
Compiling Generated Code	824
How to Modify the Stylus Studio Classpath	824
How to Compile and Run Java Code in Stylus Studio	825
Deploying Generated Code	826
Chapter 11: Composing Web Service Calls	827
Overview	828
How to Compose a Web Service Call	828
Obtaining WSDL URLs	831
Browsing UDDI Registries	831
How to Browse UDDI Registries	833
Modifying a SOAP Request	835
Understanding Parameters	835
Displaying a WSDL Document	836
How to Modify a SOAP Request	837
Testing a Web Service	837
What Happens When You Test a Web Service	837
Other Options for Testing a Web Service	837
How to Test a Web Service	838
Saving a Web Service Call	839
Using Web Service Calls as XML	839
How to Save a Web Service Call	841
Generating a Java Web Service Client	842
About the Generated Code	842
The Package Name	842
Target Directory	842
Project Folders	842
Classes	842
How to Generate a Java Web Service Client	843
Generating XQuery from a Web Service Call	844
Example	844
What Happens When You Generate XQuery	845
How to Generate XQuery from a Web Service Call	845

Creating a Web Service Call Scenario	846
Overview of Scenario Features	846
Scenario Names	847
Transport Protocol and Client Settings	847
Other Transport Settings	847
How to Create a Scenario	849
How to Run a Scenario	850
How to Clone a Scenario	850
Chapter 12: Building XML Pipelines	853
What is an XML Pipeline?.	854
Example of an XML Pipeline in Stylus Studio	854
XML Pipeline Terminology	855
XML Pipeline Semantics	856
The XML Pipeline Editor	857
Parts of the XML Pipeline Editor	858
XML Pipeline Editor Toolbar	859
Menu Actions	860
Steps for Building an XML Pipeline	860
Planning an XML Pipeline	861
Design Approaches	861
Understand the Requirements	862
Bottom-Up Design	862
Top-Down Design	863
XML Pipeline Components	864
Transformations	864
Flow Control	865
Data Sources	865
Input and Output Ports	866
Identifying Resources	866
Deployment Considerations	867
Use Case: Building order.pipeline	867
order.pipeline Requirements	868
Getting Started: Creating a New XML Pipeline	869
Save the XML Pipeline	869
XML Pipeline Scenarios	870
Specifying an Execution Framework	870
When to Specify the Execution Framework	871

Contents

Configuring Data Sources	871
Ways to Configure Non-XML Data Sources	871
Convert booksXML.txt Using a Built-in XML Converter	872
Create a ConvertToXML Node for booksXML.txt	873
Create a ConvertToXML Node for order.edi	875
Renaming Nodes	875
The XML Pipeline So Far	876
Using XQuery to Merge Source File Data	876
Using Variables to Reference Data Sources	876
Looking at the XQuery Code	879
Adding an XQuery Node	881
Changes to Source Documents	882
Setting the XQuery Node Data Sources	882
Default and Specified Port Values	883
An Alternate Way to Create ConvertToXML Nodes	883
Testing the XML Pipeline	883
Setting a Value for an Output Port	883
Designing a Report from the XML Document	885
Adding XSLT and XQuery Transformations	887
Add createReport.xsl	887
Add createReport.xquery	889
Finishing Up	892
Working with Nodes	893
Types of Nodes	893
Adding Nodes to an XML Pipeline	893
Using Existing Documents	894
Using the Toolbox	894
Node and Port Names	895
XQuery and XSLT Nodes	896
Input Ports	896
Output Ports	897
Scenario Properties	897
Changes to Source Code	897
Managing Processor Conflicts	897
XSL-FO Nodes	898
Input Port	899
Output Ports	899
Pipeline and Related Nodes	899
Example	899

Pipeline Node Input and Output Ports	900
How to Include an XML Pipeline.	901
Validate Nodes	901
Using Multiple XML Schemas.	902
Input Port	902
Output Ports	903
Choose Nodes	903
Input Ports.	903
Adding Input Ports	904
Output Ports	904
Adding Output Ports	905
ConvertToXML and ConvertFromXML Nodes	905
Specifying an XML Converter URL	905
Creating a ConvertToXML Node.	906
Input Port	907
Output Ports	907
For More Information	907
Stop and Warning Nodes.	907
Stop Nodes	907
Warning Nodes	908
XML Parser Nodes	909
Input Port	909
Output Ports	909
XML Serializer Nodes.	910
Input Port	910
Output Ports	910
Working with the XML Pipeline Diagram	911
Displaying a Grid	911
Labeling.	911
Zoom	912
Edge Style	912
Manipulating Nodes in the Diagram.	914
Saving the XML Pipeline Diagram as an Image.	914
Labeling XML Pipeline Diagrams	915
Debugging an XML Pipeline.	916
Cross-Language Debugging	916
Execution Framework Determines Debugging Support	917
Setting and Removing Breakpoints.	917
Running the Debugger.	918

Contents

Stepping Into a Node	919
Stopping Debug Processing	920
Generating Code for an XML Pipeline	920
Processor Settings and Code Generation	920
Processors for which Code Generation is Supported	920
Code Generation Settings	921
How to Generate Java Code for an XML Pipeline	922
Compiling Generated Code	922
How to Compile and Run Java Code in Stylus Studio	923
Troubleshooting Compiling Inside Stylus Studio	923
Compiling Java Code Outside Stylus Studio	924
Running Java Code in Stylus Studio	924
Deploying Generated Code	924
XML Pipeline Node Properties Reference	924
Choose Node Properties	925
Input Port	925
Node	926
Output Port	926
ConvertFromXML Node Properties	927
Input Port	927
Node	927
Output Port	927
ConvertToXML Node Properties	928
Input Port	928
Node	928
Output Port	928
Pipeline Node Properties	929
Input Port	929
Node	929
Output Port	929
Pipeline Input Node Properties	930
Node	930
Output Port	930
Pipeline Output Node Properties	931
Input Port	931
Node	931
Stop Node Properties	931
Input Port	931
Node	932

Validate Node Properties	932
Input Port	932
Node	932
Output Port	933
Warning Node Properties	933
Input Port	933
Node	933
Output Port	934
XML Parser Node Properties	934
Input Port	934
Node	934
Output Port	935
XML Serializer Node Properties	935
Input Port	935
Node	935
Output Port	936
XQuery Node Properties	936
Input Port	936
Node	936
Output Port	937
XSL-FO Node Properties	937
Input Port	937
Node	937
Output Port	938
XSLT Node Properties	938
Input Port	938
Node	938
Output Port	939

Chapter 13: Publishing XML Data 941

The Stylus Studio XML Publisher	942
Parts of the XML Publisher Editor	942
Building an XML Publisher Report	943
Process Summary	943
How to Create an XML Publisher Report	944
The XML Publisher Canvas	944
Working with Data Sources	945
How Data Sources are Represented in XML Publisher	945
Working with Namespaces	946

Contents

Adding a Data Source	946
Specifying a Default Data Source	947
Data Source Required for XSLT	948
Using XML Schema or DTD as a Data Source	948
Choosing a Root Element	949
Associating an XML Instance with the Schema	949
Grouping Data	950
What is a Relationship?	950
Creating a Relationship	951
Example – Using a Relationship in a Report	954
Deleting a Relationship	959
Adding Data to a Report	959
How to Add Data to a Report	959
Example: Dropping a Repeating Node	960
How Data is Represented on the Canvas	961
Example	961
More About the Navigation Bar	963
Click the Glyph to Navigate	964
Working with Report Components	966
Types of Components	966
Tables	967
Creating a Table	968
Graphical Representation	968
Sorting	969
Adding Rows and Columns	969
Deleting Rows, Columns, and Tables	969
Lists	970
Creating a List	970
Graphical Representation	971
Sorting	971
Adding Items	971
Deleting an Item or a List	971
Text	972
Creating a Text Component	972
Graphical Representation	972
Images	973
Creating an Image	973
Graphical Representation	973
Specifying an Image Source	974

Specifying Image Size	976
Repeaters	976
Creating a Repeater.	976
Graphical Representation	976
Sorting.	977
Ifs.	977
Creating an If	977
Graphical Representation	978
Example	978
Component Properties	980
Context and XPath Sub-Properties	980
The Properties Window	980
Example: Using Context and XPath Sub-Properties to Format Text	981
Entering XPath Expressions	983
Formatting Components	984
Formats	984
Ways to Apply Formats	984
Formatting Influenced by Component Hierarchy.	985
Setting Default Properties.	988
Clearing Formats.	988
Generating Code for an XML Publisher Report	989
Supported Languages and Documentation Types	989
Sources	990
Additional Sources	990
More About Relational Sources	991
How to Generate Code	992
Example: Building an XML Publisher Report	993
Getting Started.	993
Insert and Populate a Table	993
Simple Table Formatting.	995
Format Data Conditionally	996
Generate the Code	998
Properties Reference	999
Context and XPath Sub-Properties	1000
Body Properties	1000
Table Properties.	1001
Row Properties	1002
Column Properties	1002
Cell Properties.	1002

Contents

List Properties	1003
Item Properties	1003
Text Properties	1003
Repeater Properties	1004
If Properties	1005
Image Properties	1005
Chapter 14: Integrating with Third-Party File Systems	1007
Using Stylus Studio with TigerLogic XDMS	1007
Overview	1008
TigerLogic XDMS Version Support	1008
Connecting to TigerLogic XDMS	1009
What Happens When You Connect	1009
How to Connect to TigerLogic XDMS	1009
Reconnecting	1010
Using Documents Stored on TigerLogic XDMS	1011
Opening Documents	1011
Saving Documents	1011
Using Documents in XQuery and XSLT	1011
Creating Collections	1012
Chapter 15: Extending Stylus Studio	1013
Custom XML Validation Engines	1014
Registering a Custom Validation Engine	1014
Configuring a Custom Validation Engine	1015
The Custom Validation Engines Page	1016
How to Configure a Custom Validation Engine	1018
Custom Document Wizards	1020
Registering a Custom Document Wizard	1021
Configuring a Custom Document Wizard	1021
The Custom Document Wizards Page	1022
Defining Arguments	1026
How to Configure a Custom Document Wizard	1030
Custom File Systems	1031
Creating a Custom File System	1031
File System Interfaces	1032
Examples	1032
Registering a Custom File System	1033

The Custom File Systems Page	1033
How to Display	1034
Fields	1034
How to Register a Custom File System	1035
Chapter 16: The Stylus Studio Java API	1037
Index	1039

Preface

This Preface contains the following sections:

- [About This Manual](#) describes this manual and its intended audience.
- [Conventions in This Manual](#) describes the text formatting, syntax notation, and flags used in this manual.
- [Available Documentation](#) describes the printed and online documentation that accompanies Stylus Studio® 2007.
- [Technical Support](#) provides information on contacting Technical Support.

About This Manual

This manual describes how to use Stylus Studio 2007 to develop XML applications. It is assumed that you are familiar with XML and the concepts of it and its related technologies.

This manual has the following chapters:

- [Chapter 1, “Getting Started with Stylus Studio® 2007,”](#) provides step-by-step instructions for editing an XML document, applying a stylesheet, creating a dynamic Web page, debugging stylesheets and Java files, and mapping an XML document with one schema to an XML document with another schema.
- [Chapter 2, “Editing and Querying XML,”](#) describes how to update an XML document in the text, tree, schema, and grid views of the XML editor. It also provides information about how to query documents and handle query results.
- [Chapter 3, “Converting Non-XML Files to XML,”](#) describes how to use DataDirect XML Converters™ and Stylus Studio custom XML converters to convert files (EDI, CSV, binary, and others) to XML.

- [Chapter 4, “Working with XSLT,”](#) includes a tutorial for using XSLT and understanding how XSLT works. It provides information and instructions for using the XSLT editor to create, modify, and apply stylesheets. It also contains reference information for the various XSLT instructions you can specify in a stylesheet.
- [Chapter 5, “Creating XSLT Using the XSLT Mapper,”](#) describes how to use the Stylus Studio XML mapper. The XML mapper generates a stylesheet for transforming an XML document that uses one schema to an XML document that uses another schema.
- [Chapter 6, “Debugging Stylesheets,”](#) describes how to use the Stylus Studio debugging features.
- [Chapter 7, “Defining XML Schemas,”](#) provides information and instructions for creating and editing DTDs and XML Schema documents.
- [Chapter 8, “Defining Document Type Definitions,”](#) provides information about how to use the Stylus Studio Document Type Definition (DTD) editor to define a DTD.
- [Chapter 9, “Writing XPath Expressions,”](#) includes complete information about how to define a query, which must be an XPath expression. In addition to explicitly running a query on an XML document, you specify queries as the values of `select` and `match` attributes in stylesheets.
- [Chapter 10, “Working with XQuery in Stylus Studio,”](#) describes how to work with XQuery in Stylus Studio, including how to use the XQuery debugger.
- [Chapter 11, “Composing Web Service Calls,”](#) describes how to design, compose, and test a Web service call without writing any code, and how to use the Web service calls you create elsewhere in Stylus Studio.
- [Chapter 12, “Building XML Pipelines,”](#) describes how to use Stylus Studio to create an XML pipeline application that chains together two or more XML transformations, and how to generate Java code you can use to deploy that application.
- [Chapter 13, “Publishing XML Data,”](#) describes how to use Stylus Studio to use XML Publisher to generate XQuery or XSLT that creates HTML+CSS or XSL-FO reports using XML and non-XML data sources.
- [Chapter 14, “Integrating with Third-Party File Systems,”](#) describes how Stylus Studio is integrated with third-party file systems like RainingData[®] TigerLogic[®] XML Data Management Server (TigerLogic XDMS).
- [Chapter 15, “Extending Stylus Studio,”](#) provides a description of advanced Stylus Studio features, including information about using the Stylus Studio custom document wizard.

- [Chapter 16, “The Stylus Studio Java API,”](#) has been deprecated in Stylus Studio 2007 XML Enterprise Suite Release 2. The functionality provided by the Stylus Studio Java API has been replaced by DataDirect XML Converters™ standalone components for Java™ and .NET. See the DataDirect XML Converters documentation for more information: [http://www.xmlconverters.com/doc/..](http://www.xmlconverters.com/doc/)

Conventions in This Manual

This section describes the typographical and formatting conventions used in this manual for text, notes, warnings, and important messages.

Typographical Conventions

This manual uses the following typographical conventions:

- **Bold typeface in this font** indicates keyboard key names (such as **Tab** or **Enter**) and the names of windows, menu commands, buttons, and other user-interface elements. For example, “From the **File** menu, select **Open**.”
- *Italic text* emphasizes new terms when they are introduced.
- Code samples appear in text like this:

```
-Xdebug -Xnoagent -Xrunjdp:transport=dt_socket,  
server=y,suspend=n,address=8000 -Djava.compiler=NONE
```

- Monospace typeface indicates text that might appear on a computer screen such as
 - Code that the user must enter
 - System output (such as responses, error messages, and so on)
 - Filenames and pathnames
 - Software component names, such as class and method names

Essentially, monospace typeface indicates anything that the computer is “saying,” or that must be entered into the computer in a language that the computer “understands.”

Bold monospace typeface emphasizes text that would otherwise appear in monospace typeface.

Monospace typeface in italics or ***Bold monospace typeface in italics*** (depending on context) indicates variables or placeholders for values you supply or that might vary from one case to another.

◆ **Procedures are introduced this way:**

Syntax Notation

This manual uses the following syntax notation conventions:

- Brackets ([]) in syntax statements indicate parameters that are optional.
- Braces ({ }) indicate that one (and only one) of the enclosed items is required. A vertical bar (|) separates the alternative selections.
- Ellipses (. . .) indicate that you can choose one or more of the preceding items.

Information Alerts

This manual highlights special kinds of information by shading the information area, and indicating the type of alert in the left margin.

Tip A **Tip** flag identifies information that can help you use Stylus Studio more effectively – short-cuts, alternatives, and information about system behavior are all examples of tips.

Note A **Note** flag indicates information that complements the main text flow. Such information is especially needed to understand the concept or procedure being discussed.

Important An **Important** flag indicates information that must be acted upon within the given context in order for the procedure or task (or other) to be successfully completed.

Warning A **Warning** flag indicates information that can cause loss of data or other damage if ignored.

Edition Alerts

Not all features are supported in all editions of Stylus Studio. Documentation that describes features peculiar to a given edition is identified with an alert like the following:



XML Pipelines are available only in Stylus Studio XML Enterprise Suite.

See [Stylus Studio Editions](#) on page 3 for more information about the features that are available in a given edition.

Video Alerts

Stylus Studio provides dozens of video demonstrations of editing tools and features for XSLT, XQuery, XML Schema, relational-toXML conversion, and others. Sections in the documentation that describe a Stylus Studio feature for which a video demonstration exists include an alert like the following:



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch a video on Custom XML Conversions](#).

Clicking either the television icon or the hyperlink launches the video.

Video Descriptions

You can find descriptions of all Stylus Studio video demonstrations here: http://www.StylusStudio.com/xml_videos.html

Available Documentation

Table 1 lists the documentation supplied with Stylus Studio. In addition to the documentation listed in this table, Stylus Studio comes with sample files. All documentation is included with the Stylus Studio media and downloads.

Table 1. The Stylus Studio Documentation Set

Document	Description
<i>Stylus Studio® 2007 User Guide</i>	Describes how to use Stylus Studio to develop XML applications using XML, SQL/XML, XQuery, Web Services, and XSLT.
<i>ReadMe</i>	Describes features in the current release of Stylus Studio plus late-breaking information and known issues. The release notes are located in the \doc directory where you installed Stylus Studio.
Online help	Stylus Studio's online help system can be accessed from the application by pressing F1 or by selecting Help > Documentation from the menu bar. You can also view the help independently of the application, by opening <i>ide.chm</i> in the \doc directory where you installed Stylus Studio.

Technical Support

Submit questions and report problems using the [Stylus Studio Developer Network \(SSDN\)](#). SSDN has numerous active forums, including specialized forums for

- XQuery
- XSLT
- Code samples and utilities
- General technical questions
- Feature requests

SSDN is fully searchable, and contains current as well as historical information about Stylus Studio and XML technologies. If you cannot find the answer to the question you have, submit it to the forum and a Stylus Studio technician will respond to you.

When submitting a question or reporting an issue, it often helps to state the version of Stylus Studio you are running (click **Help > About Stylus Studio** on the menu bar) as well as any other information about your environment you think might be relevant (such as the JVM version you are using, for example).

Chapter 1 Getting Started with Stylus Studio® 2007

Stylus Studio® 2007 (Stylus Studio) is an integrated development environment (IDE) for XML and related technologies. Stylus Studio allows you to design, develop, and test XML applications using its intuitive graphical interface, textual editors, and debuggers for XML, XML Schema, DTD, XQuery, XSLT, Web services, and Java.

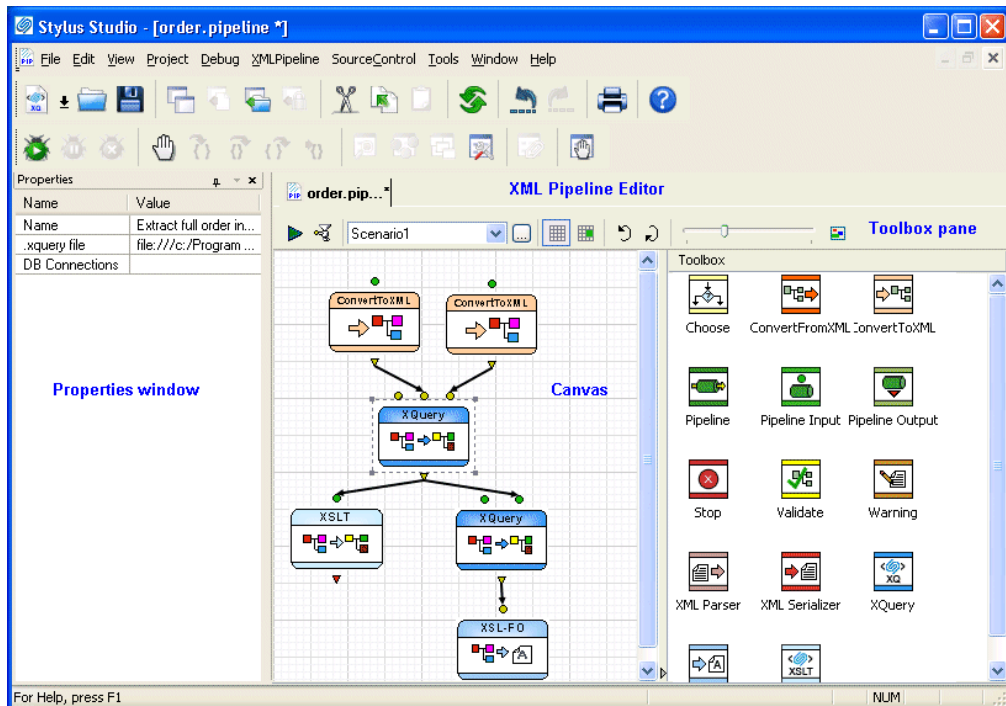


Figure 1. Stylus Studio's XML Pipeline Editor

Stylus Studio includes modules for:

- XML
- XQuery
- XSLT
- XML Pipelines
- XML reporting
- Relational data sources
- DTD
- XML Schema
- Web services
- Java
- Converting non-XML files to XML, and vice versa

Each module has one or more editors to help you author, edit, and debug XML applications.

This chapter provides a tour of the basic operations Stylus Studio provides with each of its modules. It also includes information about opening files in any module, using projects to organize files, and setting options that affect all modules.

This chapter is organized as follows:

- [“Stylus Studio Editions”](#) on page 3
- [“Starting Stylus Studio”](#) on page 5
- [“Updating an XML Document – Getting Started”](#) on page 6
- [“Working with Stylesheets – Getting Started”](#) on page 27
- [“Using the XSLT Mapper – Getting Started”](#) on page 47
- [“Debugging Stylesheets – Getting Started”](#) on page 55
- [“Defining a DTD – Getting Started”](#) on page 63
- [“Defining an XML Schema Using the Diagram Tab – Getting Started”](#) on page 68
- [“Opening Files in Stylus Studio”](#) on page 93
- [“Working with Projects”](#) on page 101
- [“Customizing Tool Bars”](#) on page 119
- [“Specifying Stylus Studio Options”](#) on page 121
- [“Defining Keyboard Shortcuts”](#) on page 125
- [“Using Stylus Studio from the Command Line”](#) on page 127

- [“Managing Stylus Studio Performance”](#) on page 130
- [“Configuring Java Components”](#) on page 134

Stylus Studio Editions

Stylus Studio is offered in several editions to provide a tool that is appropriate for every level of user, from integration architects and application developers providing enterprise-class solutions, to students and non-professional users just starting out with XML and related technologies.

Stylus Studio XML Enterprise Suite

Stylus Studio XML Enterprise Suite is Stylus Studio’s most comprehensive XML IDE, offering a complete and robust set of tools for writing, testing, debugging, and deploying XML applications. In addition to editors for XML, XML Schema, XQuery, and XSLT, Stylus Studio XML Enterprise Suite provides the following features exclusive to this edition of Stylus Studio:

- Java code generation for deployment of XQuery and XSLT transformations
- The ability to expose XML Schema to Java binding
- XQuery and XSLT Profilers to help you gather performance metrics and troubleshoot performance bottlenecks
- Custom XML Conversions, a module that allows you to easily convert non-XML files to XML
- Web Service Call Composer to help you build and test calls to hundreds of industry-standard Web services
- Integration with Raining Data Tiger Logic XDMS
- Support for OASIS catalogs, including dozens of catalogs bundled with Stylus Studio

Stylus Studio XML Professional Suite

Stylus Studio XML Professional Suite provides a complete set of tools for the XML application developer, including

- XML Differencing for comparing multiple XML documents and folders
- Custom file systems

These features are also included in Stylus Studio XML Enterprise Suite.

Stylus Studio Home Edition

Stylus Studio Home Edition is a value-priced XML IDE that provides an excellent way to learn about and work with XML and its related technologies. Stylus Studio Home Edition offers many of the features of Stylus Studio XML Professional Suite, allowing you to do real work with XML, XML Schema, XSLT, DTD, and other important XML technologies.

Edition Alerts

The *Stylus Studio® 2007 User Guide* describes features found in all Stylus Studio editions. Alerts, like the one shown here, are used to identify documentation describing features found only in particular Stylus Studio editions.



The XML Editor **Grid** tab is available only in Stylus Studio XML Enterprise Suite and Stylus Studio XML Professional Suite.

More Information

For a complete description of Stylus Studio XML Enterprise Suite, Stylus Studio XML Professional Suite, and Stylus Studio Home editions, see the Stylus Studio Web site:
http://www.StylusStudio.com/xml_feature_comparison.html

Starting Stylus Studio

Throughout this chapter, you perform exercises that require you to first start Stylus Studio. For example, if you installed Stylus Studio XML Enterprise Suite, you would

- ◆ **Select Start > Programs > Stylus Studio XML Enterprise Suite > Stylus Studio.**

The path shown here assumes that you accepted the defaults when you installed Stylus Studio. If you did not, you must alter your selection path accordingly.

You can also start Stylus Studio by double-clicking the desktop icon, which is added to your desktop by default when you install Stylus Studio:



Figure 2. Example of a Stylus Studio Desktop Icon

On startup, Stylus Studio displays the **Tip of the Day** dialog box.

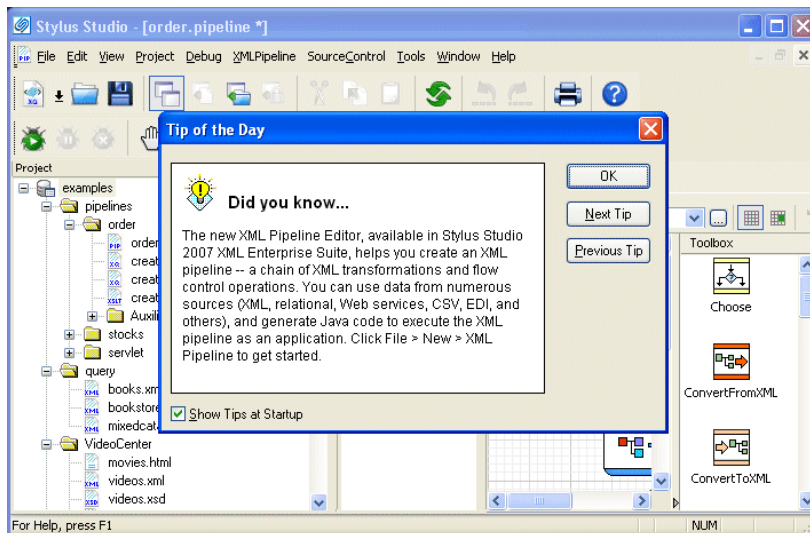


Figure 3. Stylus Studio on Startup (XML Professional Suite Shown)

Getting Updates

By default, Stylus Studio checks the Stylus Studio Web site for newer versions each time you start the application. You can review and modify this and other application settings by selecting **Tools > Options** from the menu bar and selecting the **Application Settings** page.

If you want, you can perform this check manually by selecting **Help > Check for latest version** from the Stylus Studio menu.

Getting Help

As you use Stylus Studio, you can press F1 at any time to obtain context-sensitive help. If you want, you can open the online help manually (and independent of the Stylus Studio application) by selecting **Start > Programs > Stylus Studio XML Edition Name > Stylus Studio Documentation**.

Note The online documentation is not installed with Stylus Studio. The first time you access the online documentation, you are prompted to download it from the Stylus Studio web site. By default, the online documentation is installed in the \doc directory where you installed Stylus Studio.

Updating an XML Document – Getting Started

Each of the following topics contains instructions for editing a sample XML document. You should perform the steps in each topic before you move on to the next topic. After the first topic, some steps depend on actions you performed in a previous topic. This introduction to updating XML documents in Stylus Studio is organized as follows:

- “Opening a Sample XML Document” on page 7
- “Updating the Text of a Sample Document” on page 8
- “Updating the Schema of a Sample Document” on page 15
- “Updating the Tree Representation of a Sample Document” on page 21
- “Updating a Sample Document Using the Grid Tab” on page 24

Opening a Sample XML Document

◆ **To open the** `your-quotes.xml` **sample XML document in Stylus Studio:**

1. In the **File Explorer** window, navigate to the `examples\quotes` directory in your Stylus Studio installation directory.

Tip The `\examples` directory is a sibling of `\bin`.

2. Double-click **your-quotes.xml**.

Stylus Studio displays the `your-quotes.xml` document in the XML editor. The initial view of the document is the **Text** view, as you can see by the tab at the bottom of the window.

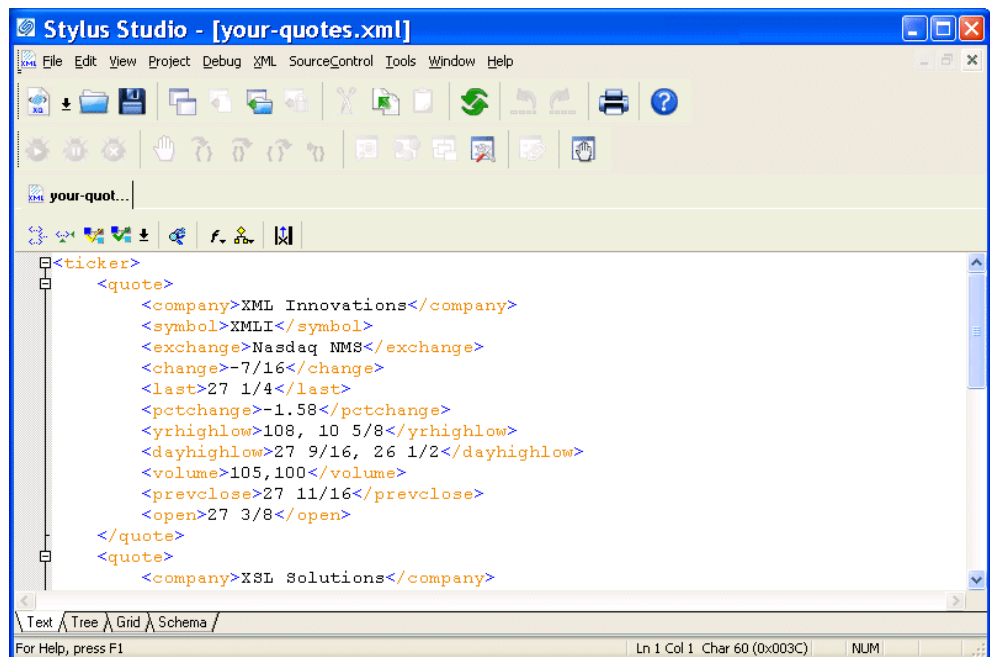



Figure 4. Editors Use Color-Keyed Text

Tip Stylus Studio uses different colors to distinguish markup, tag names, and data in all of its text editors. Orange, for example, identifies elements that are not associated with a schema. You can change the colors for editors individually. Select **Tools > Options** from the menu bar, then select **Editor Format**. You select the editor whose settings you want to modify using the **Editor** drop-down list.

Alternatives

The **File Explorer** window is the primary way to open and access files in Stylus Studio, but you can also open files using:

- The **Open** dialog box, which is displayed when you select **File > Open** from the menu bar or click the **Open**  button on the tool bar, for example.
- The **Project** window, which is displayed on the left of the Stylus Studio desktop. The **Project** window shows only those files associated with Stylus Studio projects.

For more information

See [“Opening Files in Stylus Studio”](#) on page 93 to learn more about the **File Explorer** window.

See [“Working with Projects”](#) on page 101 to learn more about projects in Stylus Studio.

Updating the Text of a Sample Document

When you update an XML document in the **Text** view of the XML editor, you can use the usual editing tools, as well as tools tailored for handling XML.

Each of the following topics contains instructions for editing a sample XML document. You should perform the steps in each topic before you move on to the next topic. After the first topic, some steps depend on actions you performed in a previous topic.

For more information on editing tools and features, see [Using the Text Editor](#) on page 145.

This section provides instructions for

- [“Displaying Line Numbers”](#) on page 9
- [“Adding Elements in the Text View of a Sample Document”](#) on page 9
- [“Copying and Pasting in the Text View of a Sample Document”](#) on page 10
- [“Undoing Operations in the Text View of a Sample Document”](#) on page 11
- [“Inserting Indents in the Text View of a Sample Document”](#) on page 11
- [“Querying in the Text View of a Sample Document”](#) on page 12
- [“Deleting and Saving Queries”](#) on page 14

Displaying Line Numbers

Stylus Studio lets you optionally display line numbers in most of its editors. Line numbers provide simple, unobtrusive points of reference that can make working large or complex documents easier. Line numbers are off by default; turn them on now.

◆ To display line numbers:

1. Select **Tools > Options** from the Stylus Studio menu. Stylus Studio displays the **Options** dialog box.
2. Click **Application Settings > Editor General**.

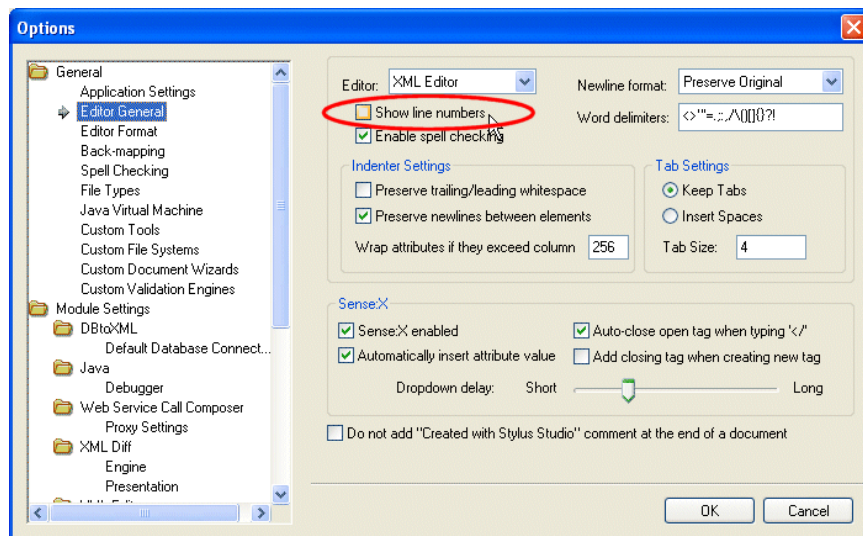


Figure 5. Sense:X and Other Editor Features are in the Options Dialog Box

3. Select **XML Editor** from the **Editor** drop-down list.
4. Click **Show line numbers**.
5. Click **OK**.

Adding Elements in the Text View of a Sample Document

◆ To add elements in the Text view of your-quotes.xml:

1. In the XML editor window, click in the first line just after `<ticker>`.
2. Press Enter and type `<quote><company>data</`.

Sense:X™ Auto-Completion

As soon as you type the closing forward slash, Stylus Studio displays **company>** because it is the only element that is appropriate to close. Automatic closing of open tags is part of Stylus Studio's *Sense:X intelligent editing*. You can change this and other Sense:X options on the **Editor General** page of the **Options** dialog box – for example, you can have Stylus Studio display a list of appropriate elements, even if that list includes one only item.

This document does not have a DTD or XML Schema associated with it. But suppose for a moment that it does have an associated schema. As soon as you type <, Stylus Studio would display a pop-up list of the elements you could add at that location. You need only double-click the element you want to add.

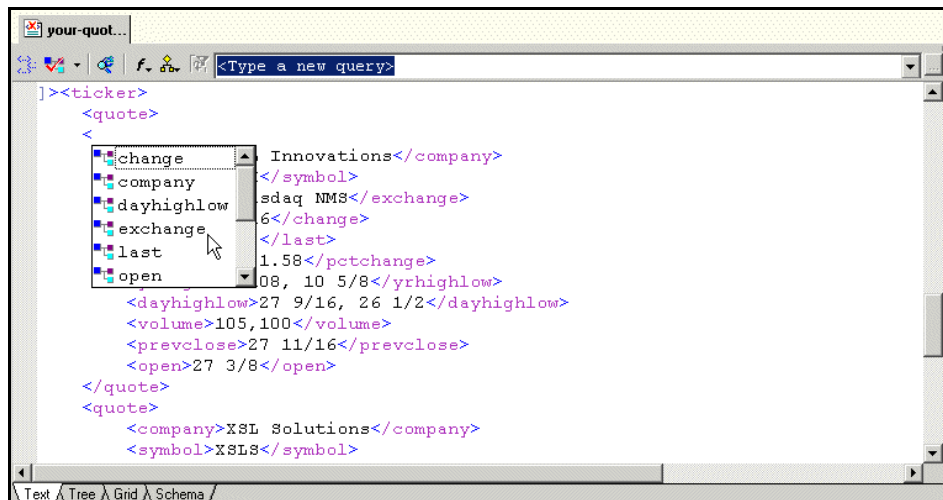



Figure 6. Example of Stylus Studio Sense:X

Copying and Pasting in the Text View of a Sample Document


- ◆ **To copy and paste elements in the Text view of your-quotes.xml:**
 1. Use the mouse to select the text for one quote element and its contents.
 2. In the menu bar, select **Edit > Copy**.
Alternatives: Press Ctrl+C or click **Copy**.
 3. Scroll down in the XML editor and click just before </ticker>.
 4. In the menu bar, select **Edit > Paste**.

Stylus Studio copies the quote element here, but the indentations are not quite right. Instructions for fixing this are in the topic [Inserting Indents in the Text View of a Sample Document](#) on page 11.

Alternatives: Press Ctrl+V or click **Paste**. .


Undoing Operations in the Text View of a Sample Document

◆ **To undo operations performed on the `your-quotes.xml` document:**

1. In the menu bar, select **Edit > Undo** to remove the text you just pasted.
Alternative: Press Ctrl+Z.
2. In the menu bar, select **Edit > Redo** to replace the text you just removed.
Alternative: Press Ctrl+Y.
3. In the XML editor window, click **Indent XML Tags** , which is the left most button. Stylus Studio displays a message that alerts you that there is an open tag for a quote element but no close tag. The messages indicates the line and column in which the error was found.
4. In the alert box, click **OK**. Because the document is not well-formed XML, Stylus Studio does not insert indents in the document. The next topic, [Inserting Indents in the Text View of a Sample Document](#) on page 11, shows how to fix the document so that it is well-formed.
5. In the menu bar, click **Edit**.
The **Undo** and **Redo** operations are no longer active. After you click the **Indent XML Tags** button, you cannot automatically undo or redo recent changes. It does not matter whether or not Stylus Studio actually inserts the indents. After you make another change, the **Undo** operation becomes active again.

Inserting Indents in the Text View of a Sample Document

◆ **To insert indents in `your-quotes.xml`:**

1. In the XML editor tool bar, click **Indent XML Tags**  again.
Stylus Studio displays the message that indicates that a close tag is missing. It specifies the element name, and the line and column numbers that identify where the error was found.

2. In the alert box, click **OK**.

Stylus Studio moves the cursor so that it appears immediately after the quote tag that has no closing tag.

Tip

The current cursor location within the document is displayed as line/column coordinates in the Stylus Studio status bar at the bottom of the Stylus Studio window.

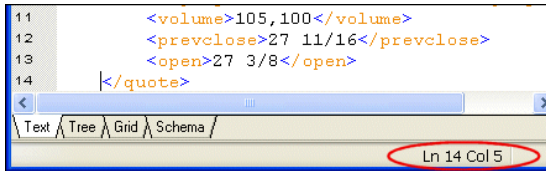




Figure 7. Document Position Displayed in Status Bar

3. In line 2, click after the </company> tag and type </>. By default, Stylus Studio displays **quote>** because it is the only element that is appropriate to close.
4. In the XML Editor tool bar, click **Indent XML Tags** . This time, Stylus Studio correctly indents the XML text.

Indent XML Tags changes your XML document by inserting white space. If this is undesirable, and you want to check for well-formedness, click the **Tree** tab at the bottom of the XML Editor window. If the document is well-formed, Stylus Studio displays the tree representation. If the document is not well formed, Stylus Studio displays a message that indicates the reason the document is not well formed and the location of the error or omission.

Querying in the Text View of a Sample Document

You use the XPath Query Editor to query XML documents using XPath. Stylus Studio supports both XPath 2.0 and 1.0. The default is XPath 2.0.

The XPath Query Editor is a dockable window that appears to the right of the XML document window. If you do not see it, click the **Show XPath Query Editor** button ().

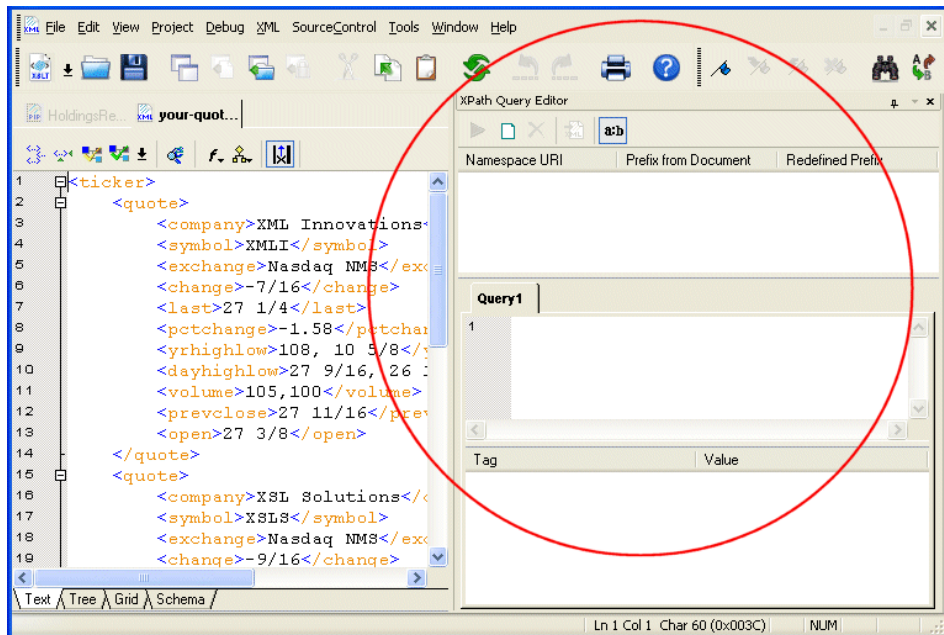



Figure 8. XPath Query Editor

◆ **To query** your-quotes.xml:

1. Click the **Query 1** tab in the XPath Query Editor.
2. Type `/ticker/quote` and click the **Execute Query** button ().
Stylus Studio runs the `/ticker/quote` query on `your-quotes.xml`, and displays the results at the bottom of the **XPath Query Editor** window.
3. In the **Query Output** window, expand the second **quote** element to view its contents.

4. Click the **symbol** element.

In the **Text** view, Stylus Studio uses its back-mapping feature to move the cursor to the source element for the **symbol** result element you clicked.

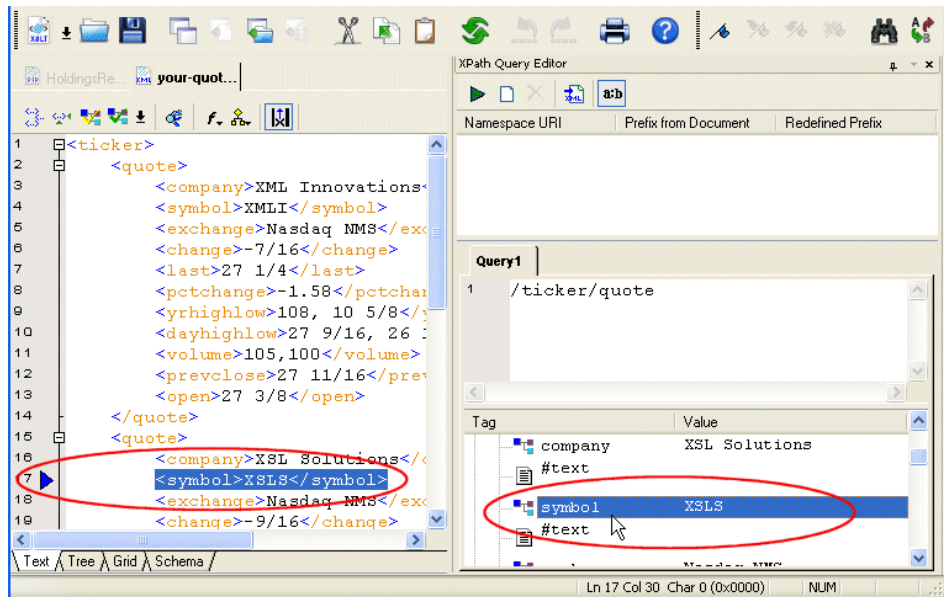




Figure 9. Backmapping from XPath Query Result to XML Document

5. In the **Text** view, click the down arrow to the right of the query field.
6. In the **XPath Query Editor** window, click the **New Query** button ()
Stylus Studio adds a new tab for each query you define.
7. Type `//company` and click the **Execute Query** button ()
Stylus Studio runs the new query and displays the results.
8. Close the **XPath Query Editor** window by clicking the **x** in that window's upper right corner.

Deleting and Saving Queries

You cannot explicitly delete a query. In addition, queries you define are not saved with an XML document unless that document belongs to a Stylus Studio *project* – if you close the XML document and then reopen it, the queries you defined in the previous editing session are no longer there.

For more information

See [“Using the XPath Query Editor”](#) on page 632 to learn more about the XPath Query Editor.

See [“Working with Projects”](#) on page 101 to learn more about projects and their role in Stylus Studio.

Updating the Schema of a Sample Document

This section provides instructions for updating the internal DTD for `your-quotes.xml`.

When an XML document has an external DTD, you can view the external DTD in the **Schema** tab of the XML Editor, but you cannot edit it. To be able to edit an external DTD, you must open it in the DTD editor. When an XML document has an internal DTD, you can view and edit it in both the **Schema** tab and the **Text** tab of the XML editor.

You should have already performed the steps in [“Updating the Text of a Sample Document”](#) on page 8. Each of the following topics contains instructions for editing the sample XML document. You should perform the steps in each topic before you move on to the next topic. After the first topic, some steps depend on actions you performed in a previous topic.

This section includes the following topics:

- [“Creating a Sample Schema”](#) on page 15
- [“Defining a Sample Element”](#) on page 17
- [“Adding an Element Reference to a Sample Schema”](#) on page 19
- [“Defining an Entity in a Sample Schema”](#) on page 20
- [“Exploring Other Features in a Sample Schema”](#) on page 20

For more information, see [“Defining a DTD – Getting Started”](#) on page 63.

Creating a Sample Schema

◆ To create the schema of a sample XML document:

1. If it is not already open, open `your-quotes.xml`. See [“Opening a Sample XML Document”](#) on page 7 if you need help with this step.
2. At the bottom of the XML editor window, click the **Schema** tab.

Stylus Studio displays the **Schema** tab, and opens the **Properties** window. The **Schema** tab displays a DTD tree, which is currently empty.

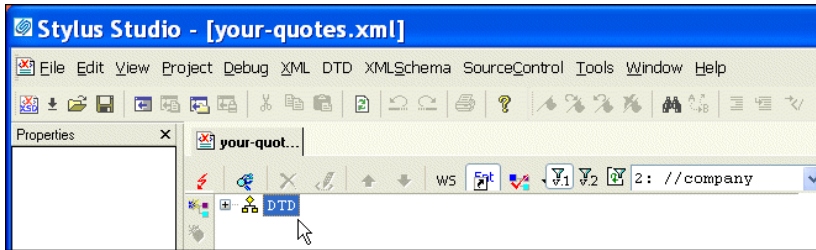


Figure 10. Default Schema Tab for Document With no Schema

3. To create a schema for your-quotes.xml, select **XML > Create Schema from XML Content** from the Stylus Studio menu.

Stylus Studio displays the **Create Schema or DTD** dialog box. By default, Stylus Studio generates an internal DTD and inserts it in a DOCTYPE element at the beginning of the document. You can also use this dialog box to generate an external XML Schema or DTD.

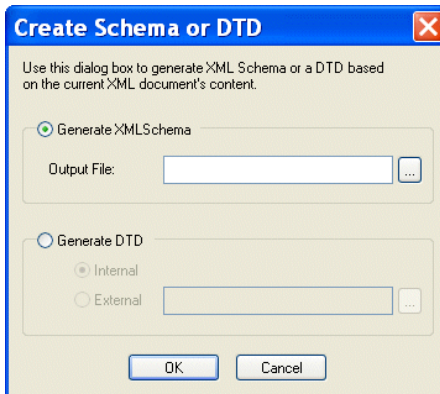


Figure 11. Create Schema or DTD Based on XML Content

4. Click **Yes** to instruct Stylus Studio to create a DTD based on the XML document content.

Stylus Studio displays a tree representation of the new, internal DTD. It also displays the **Properties** window.

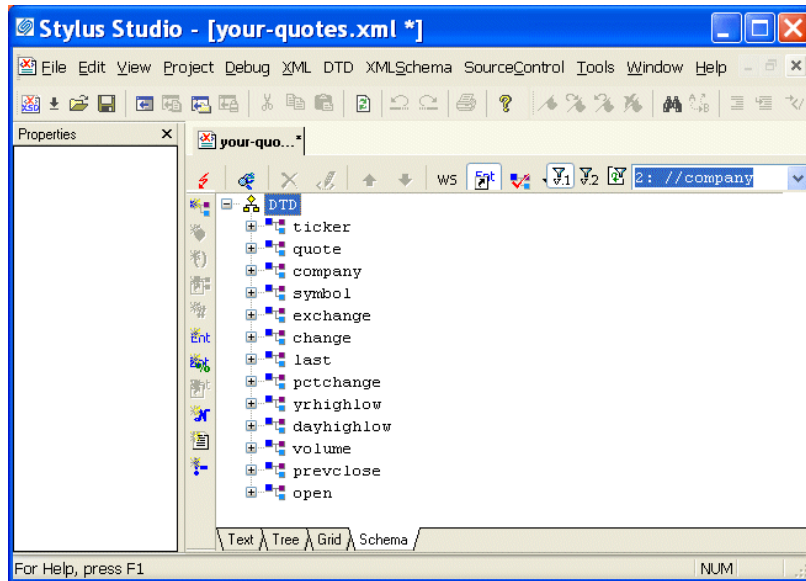


Figure 12. Result of Generating a Schema Based on XML Content

Defining a Sample Element

- ◆ **To define a new element in the Schema view of the sample schema:**

1. Click the **company** element in the **Schema** tab.

This selects the company element definition and displays the properties for the company element in the **Properties** window.

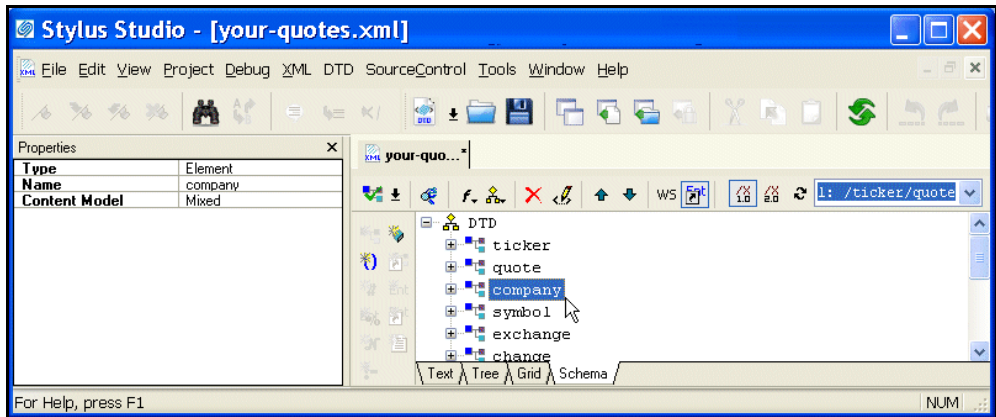


Figure 13. Properties Window Displays Element Properties

The **Content Model** property indicates the allowable contents for a company element. In this example, it is **Mixed**, which means that a company element can contain specified elements (as opposed to all elements defined in this DTD), attributes, and raw data.

Tip

Windows like the **Properties** and **Query Output** windows are docking windows – you can change their location within the Stylus Studio window, or separate them from the Stylus Studio entirely, by dragging them to the desired location.

2. Click the **DTD** node.

In the left tool bar, Stylus Studio activates only those buttons that are applicable to the DTD – you can add elements, entities, comments, and so on. But you cannot add an attribute definition, a reference to an element, or a **#PCDATA** node, for example.

3. In the left tool bar, click **New Element Definition** . Stylus Studio displays an entry field at the bottom of the tree.

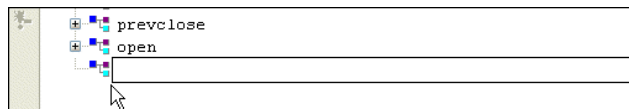



Figure 14. Entry Field for a New DTD Element

4. Type location and press Enter. Stylus Studio displays the properties for the new location element in the **Properties** window.

- In the left tool bar, click **New Modifier** . Stylus Studio displays a drop-down menu of options that specify the rules for the occurrence of the children of the new element.

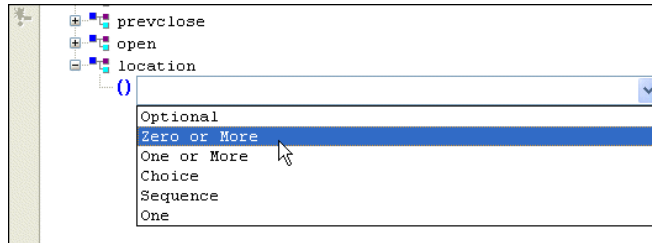






Figure 15. Drop-Down List for Modifier Values

- Double-click **Zero or More** (or click once to select it and press Enter).
- In the left tool bar, click **Add #PCDATA** . Your definition of the `location` element specifies that it can contain only raw data.


Adding an Element Reference to a Sample Schema

- ◆ **To update the definition of the `quote` element to include an optional `location` element:**

- In the **Schema** tab, expand the `quote` element.
- Click its **Sequence** modifier.
- In the left tool bar, click **New Modifier** . Stylus Studio displays an entry field for the new modifier at the end of the list of modifiers that already apply to the `Sequence` modifier. The entry field consists of a drop-down list of available values for the new modifier.
- In the drop-down list, double-click **Optional**.
- In the left tool bar, click **New Reference to Element** . Stylus Studio displays an entry field after the new **Optional** modifier.
- Type `location` and press Enter.
- To move the `location` element to be earlier in the sequence, click its **Optional** modifier.
- In the XML editor top tool bar, click **Move Up**  repeatedly until the `location` element is where you want it to be.


Defining an Entity in a Sample Schema

◆ **To define an entity in the internal DTD for your-quotes.xml:**

1. Click the **DTD** node.
2. In the left tool bar, click **New Entity** . At the end of the schema, Stylus Studio displays **Ent** and a entry field for the name of the new entity.
3. Type TCBC for the name of the entity and press Enter. In the **Properties** window, Stylus Studio displays the properties for the new entity.
4. In the **Properties** window, double-click the **Value** field.
5. Type The Country's Best Computer Company and press Enter.

Exploring Other Features in a Sample Schema

◆ **To toggle white space or validate your document:**

1. Click **Toggle Display of White Space**  to display nodes that represent white space in the DTD. Click the button again to hide the white space nodes.

2. Click **Validate Document** .

Stylus Studio displays a message in the **Output** window that indicates that the document is valid.

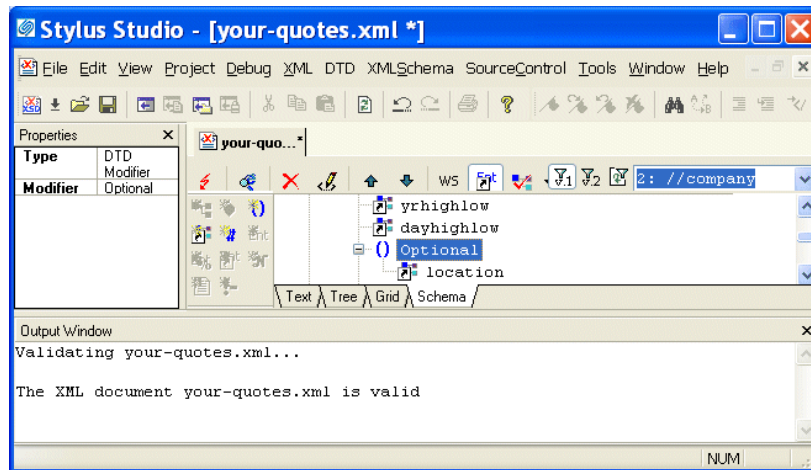


Figure 16. Output Window After Schema Validation

Updating the Tree Representation of a Sample Document

This section provides instructions for updating the DOM tree representation of the `your-quotes.xml` document.

You should have already performed the steps in [“Updating the Schema of a Sample Document”](#) on page 15. Each of the following topics contains instructions for editing the sample XML document. You should perform the steps in each topic before you move on to the next topic. After the first topic, some steps depend on actions you performed in a previous topic.

This section includes the following topics:

- [“Adding an Element to a Sample Document Tree”](#) on page 22
- [“Changing an Element’s Data in a Sample Document Tree”](#) on page 22
- [“Adding Attributes and Other Node Types to a Sample Document Tree”](#) on page 23
- [“Adding an Entity Reference to a Sample Document Tree”](#) on page 24


Adding an Element to a Sample Document Tree

◆ **To add an element to the tree representation of** `your-quotes.xml`:

1. If it is not already open, open `your-quotes.xml`.
See “Opening a Sample XML Document” on page 7 if you need help with this step.
2. At the bottom of the XML Editor window, click the **Tree** tab.
Stylus Studio closes the **Properties** window.


Tip

You can close the **Output** window if it is still open from the previous exercise.

3. Click the plus sign next to the `ticker` element to expose the children of the `ticker` element.
4. Click **New Element**  in the left tool bar to add a quote element to the document. Stylus Studio displays a drop-down menu that lists the elements you can add at that position in the tree.
5. Click **quote** and press Enter.
Stylus Studio displays a field next to the new quote element. The DTD allows a quote element to contain data.
6. Click outside the field to close it without entering data.


Changing an Element’s Data in a Sample Document Tree

◆ **In the Tree tab of** `your-quotes.xml`, **to change the data that an element contains:**

1. Expand the third **quote** element.
2. Click the **symbol** element.
3. In the XML Editor top tool bar, click **Change Value** .
Stylus Studio activates the field to the right of the **symbol** element and selects the current value.
4. In the active field, type `XSOL` and press Enter.
5. To the right of the **exchange** element, right-click **Nasdaq NMS**.
Stylus Studio displays a shortcut menu.
6. Click **Change Value**. Stylus Studio activates the value field for the exchange element.
7. In the active field, type `NYSE` and press Enter.

Adding Attributes and Other Node Types to a Sample Document Tree

◆ **To add attributes and other types of nodes to your-quotes.xml:**

1. Click the last **quote** element in the tree.
2. Click **New Attribute** .

Stylus Studio displays an attribute name field immediately below the selected quote element.

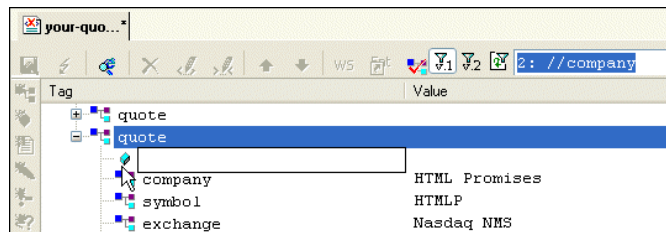






Figure 17. Adding a New Element to a Document Tree

3. In the attribute name field, type **agent** and press Enter.
Stylus Studio displays a default value for the attribute, **Text**, in an entry field to the right of the new attribute.
 4. In the attribute value field, type **Star Brokers** and press Enter.
Stylus Studio displays an entry field for a new attribute name, allowing you to easily add a number of attributes, one after the other.
 5. Click outside the attribute name field to close it.
 6. In the XML editor top tool bar, click **Validate Document** .
- Stylus Studio displays a message in the **Output** window that indicates that the document is not valid. The DTD does not specify the **agent** attribute for the **quote** element. Stylus Studio allows you to modify your document in invalid ways, which you might want to do during application design. The validation feature informs you that your document is invalid when you try to validate the document.
7. Click the **agent** attribute.
 8. In the XML Editor top tool bar, click **Delete Node** .
 9. Click **Validate Document**  again.
Stylus Studio displays a message in the **Output** window that indicates that the document is now valid.

Adding an Entity Reference to a Sample Document Tree


◆ **To add an entity reference to** `your-quotes.xml`:

1. If it is not already selected, click the **quote** element you defined in the previous topic.
2. In the left tool bar, click **New Element**  to add subelements to the new quote element.


Stylus Studio displays a drop-down menu that lists a number of elements that you can insert at this point. Scroll the list to view them all.

3. Click **company**, which is first in the list, and press Enter.

Stylus Studio displays a field next to the element name. You can enter data here, such as the name of the company. But rather than entering data, suppose you want to refer to an entity. To refer to an entity:

4. Click outside the field or press the Esc key.
5. Click **New Entity Reference** , which is the last button in the left tool bar.

Stylus Studio displays a drop-down menu that lists the defined entities.

If the **New Entity Reference** button is not active, click **Toggle Display of Entity References**  in the XML editor top tool bar. This button allows you to control whether you can refer to entities.

6. Double-click **TCBCC**.

Stylus Studio inserts the text `The Country's Best Computer Company` as the value for the `company` element.

7. Click the **Text** tab at the bottom of the XML editor window.

Stylus Studio displays the `&TBCC;` entity reference in the new `company` element.

8. Click the **Tree** tab.

Updating a Sample Document Using the Grid Tab



The XML Editor **Grid** tab is available only in Stylus Studio XML Enterprise Suite and Stylus Studio XML Professional Suite.

This section provides instructions for updating the `your-quotes.xml` document using the **Grid** tab of the XML Editor. The **Grid** tab is useful for displaying structured data. It is a

convenient way to view a document that contains multiple instances of the same type of element, for example.



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the XML Grid Editor video](#).

A complete list of the videos demonstrating Stylus Studio’s features is here: http://www.stylusstudio.com/xml_videos.html.

◆ To update an XML document using the Grid tab:

1. If it is not already open, open your-quotes.xml.
See “Opening a Sample XML Document” on page 7 if you need help with this step.
2. At the bottom of the XML Editor window, click the **Grid** tab.
Stylus Studio displays a table that contains the XML data.

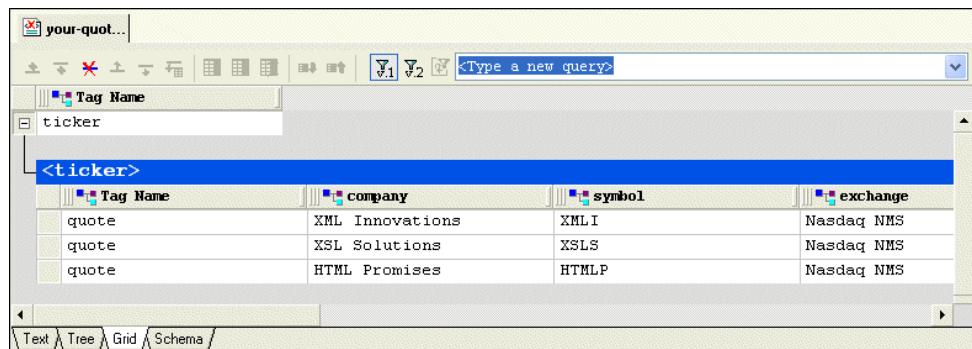


Figure 18. Grid Tab

The left most column, with the **Tag Name** heading, contains the children of the <ticker> element. The remaining columns contain the grandchildren of the <ticker> element. The heading of each column identifies the element name – company, symbol, and so on.

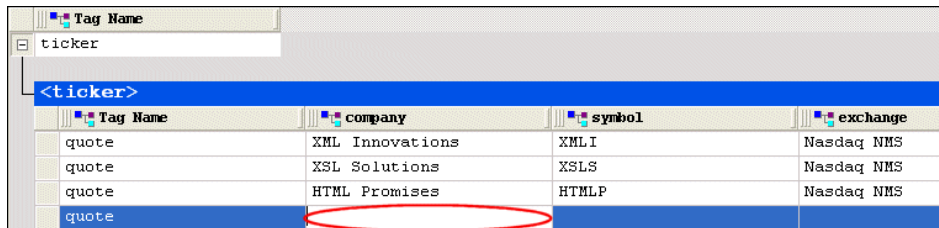
Tip

You can resize columns by dragging the handle on the column heading’s right side. You can change the element order in the document by dragging the handle on the column heading’s left side. Stylus Studio swaps positions with the column on which you come to rest.

3. Select the last row by clicking to the left of the last <quote> element.
The row is highlighted in blue.

4. Click the **Insert row after** () button.

A new instance of the <quote> element is added to the document. The cursor is placed in the <company> element cell.



Tag Name	company	symbol	exchange
quote	XML Innovations	XML I	Nasdaq NMS
quote	XSL Solutions	XSLS	Nasdaq NMS
quote	HTML Promises	HTMLP	Nasdaq NMS
quote			

Figure 19. Grid with a New Row

5. Type XML Designs and press Enter.
Stylus Studio creates the value for the <company> subelement.
6. Press Tab (or use the right arrow key) to move the cursor to the next cell in the row.
7. Repeat [Step 5](#) and [Step 6](#) to create values for the <symbol> and <exchange> subelements.
8. If you want, you can continue to add the data contained in a quote element.

Modifying Values

It is easy to change and delete values in grid fields:

- To change the value of any field, double-click the field and type the new data. Press Enter to save the change.
- To delete the value of a field, double-click the field, select the text you want to delete, and press the Delete key.

Moving Around the Grid

You can move around the grid using the mouse and the keyboard.

Using the mouse, click where you want to place the cursor.

Using the keyboard:

- Use the Tab key to advance the focus to the next cell; use Shift + Tab to move the focus to the previous cell
- Use the arrow keys to move the focus in the direction of the arrow you choose

Working with Stylesheets – Getting Started

This section helps you get started working with XSLT stylesheets. To focus on stylesheets that map XML to XML, see [“Using the XSLT Mapper – Getting Started”](#) on page 47. To learn about using XML Publisher to generate XSLT for HTML reports, see [Chapter 13, “Publishing XML Data.”](#)

Except for the first topic, each of the following topics contains instructions for working with a sample XSLT stylesheet. You should perform the steps in each topic before you move on to the next topic. After the first topic, some steps depend on actions you performed in a previous topic.

This introduction to working with stylesheets in Stylus Studio is organized as follows:

- [“Opening a Sample Stylesheet”](#) on page 27
- [“XSLT Stylesheet Editor Quick Tour”](#) on page 28
- [“XSLT Scenarios”](#) on page 32
- [“Making a Static Web Page Dynamic by Editing XSLT”](#) on page 38

To get started, you’ll need to start Stylus Studio if you haven’t already. See [“Starting Stylus Studio”](#) on page 5 if you need help with this step.

Opening a Sample Stylesheet

◆ **To open the `your-quotes.xml` sample XSLT stylesheet in Stylus Studio:**

1. In the **File Explorer** or **Open** dialog box, navigate to the `examples\quotes` directory in your Stylus Studio installation directory.

Alternative: If the Stylus Studio `examples` project is open, you can access this file from the **Project** window. To open the `examples` project, open `examples.prj` in the Stylus Studio `examples` directory.

2. Double-click **your-quotes.xml**. Stylus Studio displays the your-quotes.xml document in the **XSLT Source** tab of the XSLT editor.

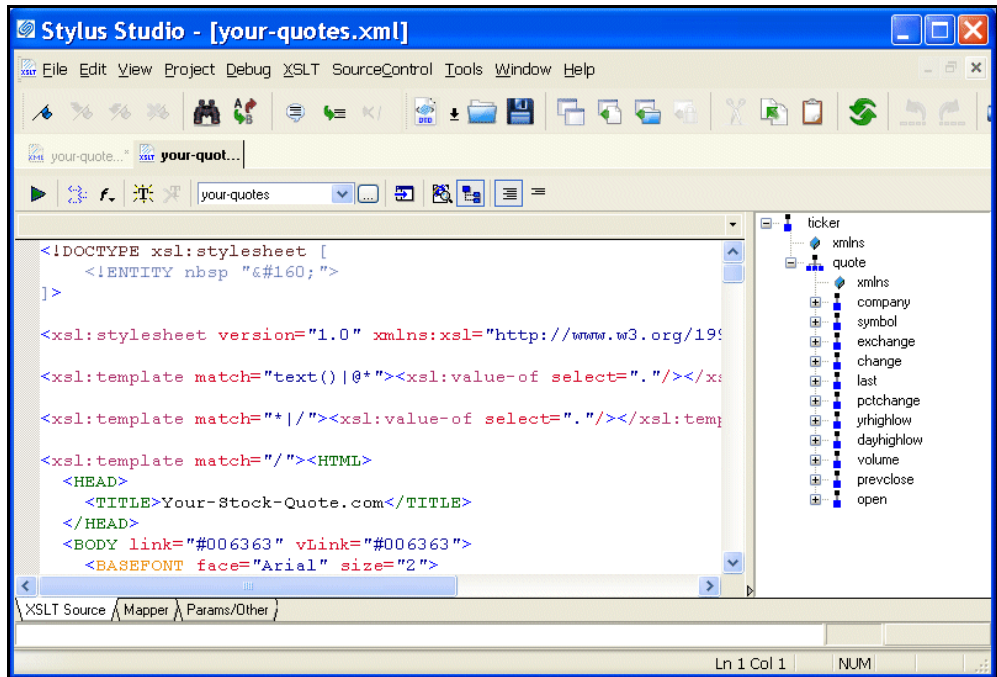


Figure 20. Stylus Studio's XSLT Editor

As with the XML Editor, Stylus Studio uses different colors to distinguish markup, tag names, and data in the XSLT Editor.

XSLT Stylesheet Editor Quick Tour

When you use the Stylus Studio XSLT stylesheet editor, you work with XSLT stylesheets, XML source documents, and result documents. This quick tour is organized to introduce you to some of the main features for working with XSLT in Stylus Studio:

- [“Parts of the XSLT Editor”](#) on page 29
- [“Exploring the XSLT Source Tab”](#) on page 29
- [“Exploring the Params/Other Tab”](#) on page 32

Parts of the XSLT Editor

The XSLT Editor consists of four tabs that allow you to work with XSLT in different ways, based on your preferences and the functionality that you desire.

- **XSLT Source.** Use the **XSLT Source** tab when you want to directly edit or view the XSLT source code that comprises your stylesheet. The XSLT Source tab can also be a good way to learn more about XSLT.

Tip XSLT source is also visible from a pane within the **Mapper** tab.

- **Mapper.** The **Mapper** tab allows you to create XSLT by graphically mapping source document nodes to nodes in a target document. Stylus Studio interprets the mappings to generate XSLT that will yield a document conforming to the document described in the **Set Target Document** pane.

Note Using the Mapper tab is discussed in detail in [“Using the XSLT Mapper – Getting Started”](#) on page 47.

- **Params/Other.** You use the **Params/Other** tab to specify the encoding Stylus Studio uses to store the stylesheet, the stylesheet’s output method, and the encoding Stylus Studio uses for the document that results from applying this stylesheet. You can also use this tab to view default values for parameters used by your stylesheet.

Exploring the XSLT Source Tab

◆ To work with the XSLT Source tab:

1. In the stylesheet text, click anywhere below the third **xsl:template** instruction (line 11).

In the status bar just below the XSLT Editor tool bar, Stylus Studio displays **match: /**. This indicates that the location you clicked is inside a template that matches the root node.

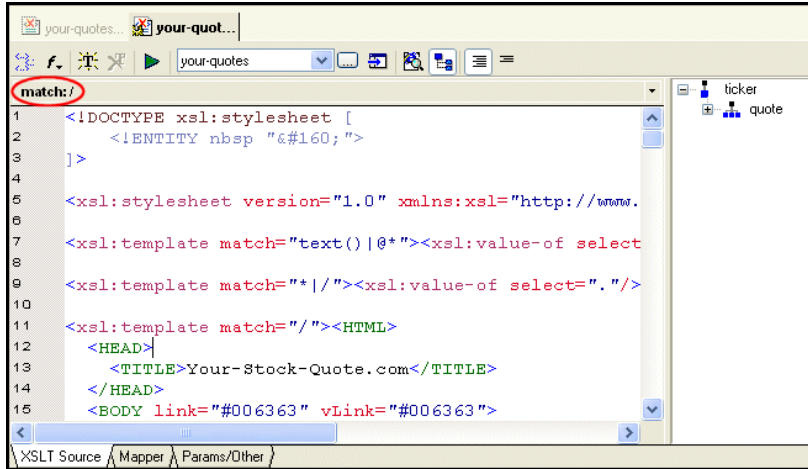



Figure 21. Current Template Identity Is Displayed at the Top of the Editor

2. Click in the **xsl:stylesheet** instruction (line 5).

Now the status bar is blank. This instruction is not part of a template.

3. In the XSLT Editor tool bar, click **Add a new template** .

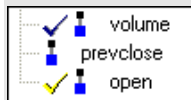
Stylus Studio inserts the following after the last template already specified in the stylesheet.

```
<xsl:template match="NewTemplate">
</xsl:template>
```

To define a new template, replace `NewTemplate` with the match pattern you want, and add contents to the new template as needed.

Tip

You can also create a new template by double-clicking a node on the schema tree. Templates that match nodes in the XSLT document are displayed with a check in the schema tree, as shown here.



Yellow indicates that the text cursor in the XSLT source is within that template.

- In the XSLT Editor tool bar, click **Template Mode** , which is the right most button. Stylus Studio displays only the new template.

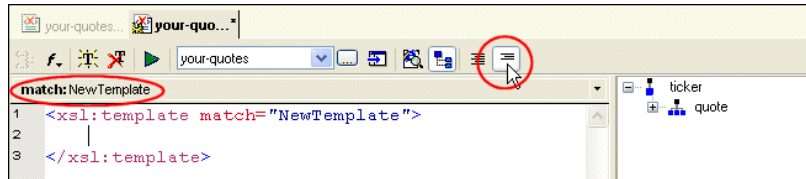


Figure 22. Use Template Mode to Focus on a Single Template

You can edit the stylesheet in either template mode or in full source mode. In template mode, Stylus Studio displays one template at a time. In full source mode, Stylus Studio displays the whole stylesheet.

Tip

In large or complex stylesheets, use the XSLT Editor’s status bar to identify the current template.

- In the upper right corner of the editing pane, click the down arrow. Stylus Studio displays a list of the templates in the stylesheet with their match patterns.

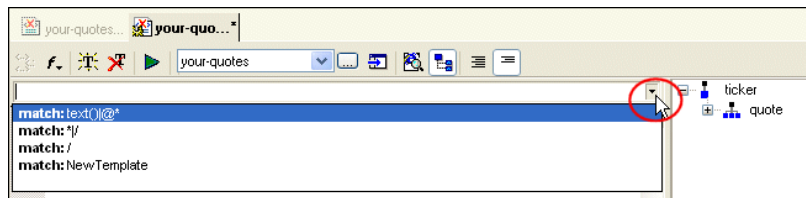



Figure 23. You Can Show Individual Templates in a Stylesheet

- Click **match: */|**. This displays the template that matches every element and the root node.

Every stylesheet that Stylus Studio creates includes two built-in templates. One built-in template matches every element and the root node. The other built-in template matches all text and attribute nodes. See [“Using Stylus Studio Default Templates”](#) on page 384.

To delete a template, click the match pattern for the template you want to delete and then click **Delete template**  in the XSLT Editor tool bar. You must be in template mode to delete a template.

7. Click **Full Source Mode** .

Stylus Studio displays the complete stylesheet. The cursor is at the beginning of the template that was being displayed in template mode.

Exploring the Params/Other Tab

◆ **Click the Params/Other tab:**

Drop-down menus let you specify the encoding format used to store the stylesheet in Stylus Studio, as well as method and encoding output attributes. A simple grid displays the name, source URL, and default value of any global parameters used by the active stylesheet, as well as by any imported ones.

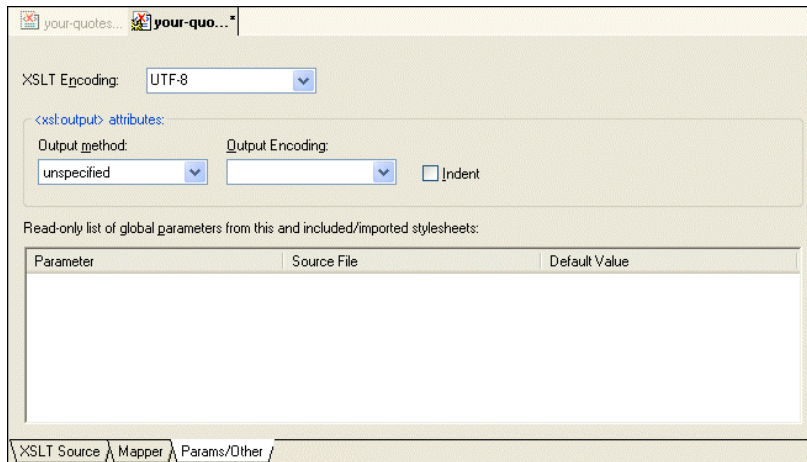


Figure 24. Specify XSLT Parameters Here or in XSLT Source

All information that you can specify in the **Params/Other** tab can also be specified in the XSLT source. For example, you can specify the XSLT encoding in the processing instruction at the beginning of the stylesheet; you can specify the output method and encoding with the `xsl:output` instruction. Stylus Studio automatically updates the XSLT source with any changes you make in the **Params/Other** tab, and vice versa.

XSLT Scenarios

To apply a stylesheet to an XML document in Stylus Studio, you use a scenario. A *scenario* is a group of customizable settings that allows you to experiment with different source XML documents (that is, the XML document to which you will apply the XSLT),

processors, parameter values, post-processors, and profiling settings. You can also use scenarios to perform validation on the XML document that results from the XSLT processing. (Validation is always performed before any post-processing you specify.)

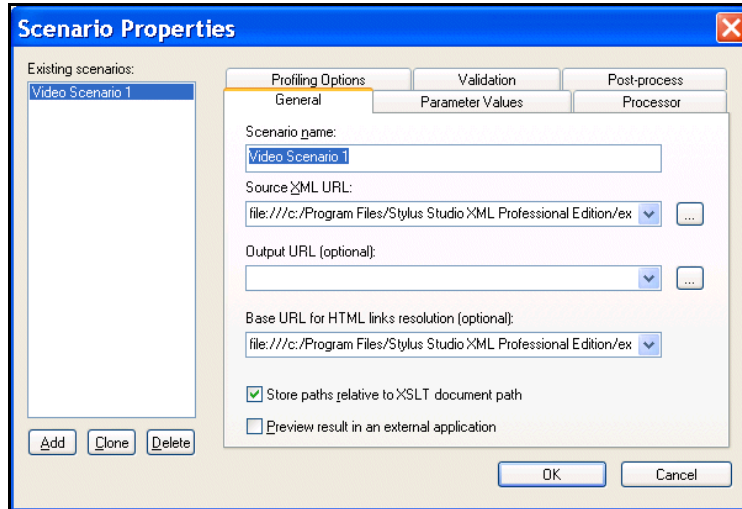


Figure 25. Scenarios Let You Easily Test Stylesheets and XML Source

You can define multiple scenarios using different settings to see how each affects document processing. Stylus Studio also supports scenarios for Web service calls, XQuery, and XML pipelines.

An XSLT scenario is defined by a single stylesheet-XML document pair. You can associate any number of scenarios with a stylesheet, though only one scenario can be in effect at the time the XSLT is processed. Similarly, you can associate any number of scenarios with an XML source document.

Tip Stylus Studio lets you work with several XSLT processors, including Saxon, MSXML and .NET.

A scenario has already been created for the your-quotes.xsl stylesheet, using the your-quotes.xml as the source XML document. Run the scenario now and look at the output created by the XSLT defined in your-quotes.xsl.

- ◆ **To run a scenario, click Preview Result** .

Stylus Studio processes the source XML document using the XSLT stylesheet you specify and displays the results in the **Preview** window.

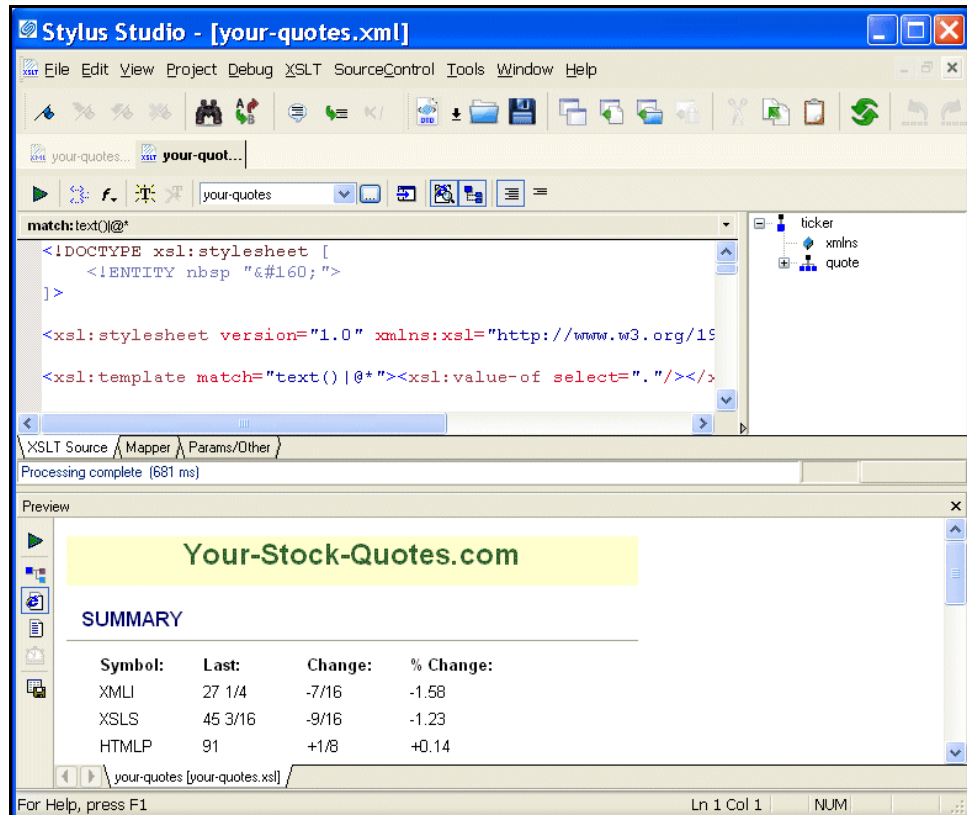




Figure 26. XSLT Processing Results are Shown in the Preview Window

By default, results are displayed using a Web browser. If you choose, you can display results in tree or text format, by clicking **Preview in Tree**  and **Preview Text**  in the **Preview** window tool bar.

Use the scroll bar to review the HTML in the **Preview** window. You can see that the values come from the XML document `your-quotes.xml`.

- Tip** If it is not already open, you can open the source XML document specified in a scenario by clicking **Open XML From Scenario**  in the XSLT Editor tool bar.

Working with Scenarios

To define additional scenarios, click the down arrow next to the scenario field in the XSLT Editor tool bar, and click **Create Scenario**. After you have more than one scenario, click the same down arrow to select the scenario you want to use to preview a result.

To change the properties of a scenario, or to delete a scenario, select the scenario you want to change or delete, and then click **Browse**  to the right of the scenario name field. Stylus Studio displays the **Scenario Properties** dialog box.

About Preview

When you preview a result, Stylus Studio automatically saves the changes you have made to the document. If you want to revert to the document's previous state, you can use the undo function (**Edit > Undo**).

Working with a Sample Result Document

◆ **To work with a sample result document, follow these steps:**

1. In the **Preview** window, click anywhere in the display.

Using its back-mapping functionality, Stylus Studio displays the template in the XSLT Editor's status bar and flags the line that generated the line you clicked with a blue pointer.

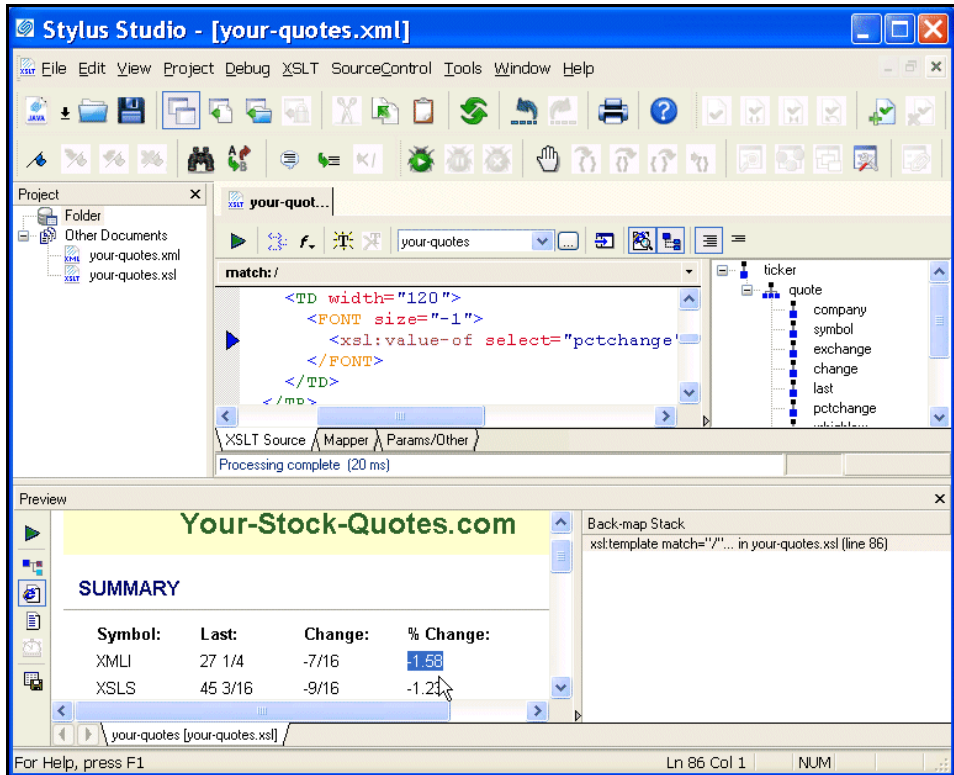



Figure 27. Back-mapping Shows which XSLT Generated a Result

- In the left tool bar of the **Preview** window, click **Preview Text** . Stylus Studio displays the HTML file that generates the browser display.

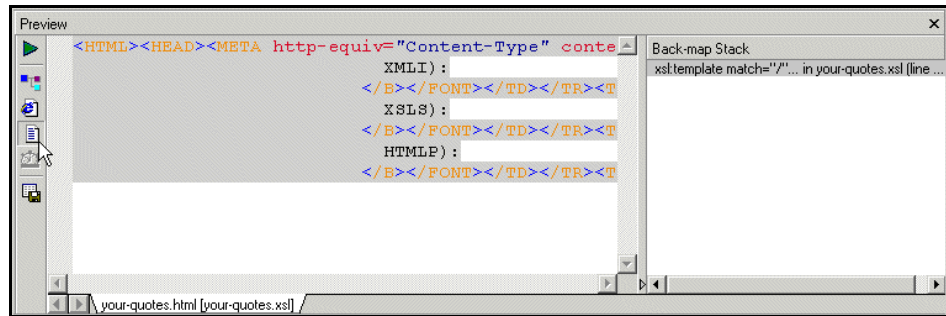



Figure 28. You Can Render XSLT Results as Plain HTML

- Click anywhere in the HTML display. The gray background identifies any HTML that was generated by the same template.
This works in reverse as well. If you click a line in a template (full source mode or template mode), Stylus Studio uses a gray background to display the HTML generated by that template.
- In the left tool bar, click **Export Preview** . Stylus Studio displays the **Save As** dialog box. If you want, you can enter a file name and click **Save** to preserve the generated HTML file. Otherwise, click **Cancel**.

Note

Notice the tab at the bottom of the **XSLT Preview** window. It specifies **your-quotes [your-quotes.xsl]**. After you create another scenario and apply the stylesheet in that scenario, another tab with the name of that scenario will be displayed. You can click the tab for the result you want to view and easily compare result documents from different scenarios.

Making a Static Web Page Dynamic by Editing XSLT

Suppose you have an XML document with a repeating element, and you want to build a Web page that displays information for each of the elements. For example, you might have a document that contains information for hundreds of books, customer accounts, sales calls, or even chemistry experiments. It is relatively easy to create an HTML file that elegantly displays the information for one element. You can then import this file into Stylus Studio, and modify it in a few steps to display all the elements in your XML document.

This section provides step-by-step instructions for converting a static Web page that displays information about one video into a dynamic Web page that displays information about many videos. The process is described in the following major steps:

- “Importing a Sample HTML File” on page 39
- “Creating the video Template” on page 41
- “Instantiating the video Template” on page 43
- “Making Titles Dynamic” on page 44
- “Making Images Dynamic” on page 45
- “Making Summaries Dynamic” on page 46

If you follow the instructions in these topics, you create the `video.xsl` stylesheet. Stylus Studio includes a sample version of this stylesheet. It is `sampleVideo.xsl`, and it is in the `examples\VideoCenter` directory of your Stylus Studio installation directory. If the Stylus Studio `examples` project is open, you can access this file from the **Project** window. To open the `examples` project, open `examples.prj` in the Stylus Studio `examples` directory.

Stylus Studio includes an additional sample stylesheet, `sample2Video.xsl`. This stylesheet performs the same function as `sampleVideo.xsl`, but it does not use the `xsl:apply-templates` instruction. Instead, it directly executes the instructions.

Importing a Sample HTML File



The HTML to XSLT Document Wizard is available only in Stylus Studio XML Enterprise Suite and Stylus Studio XML Professional Suite.

◆ To import an HTML file:

1. In the Stylus Studio menu bar, click **File > Document Wizards**.
2. Click the **XSLT Editor** tab.
3. Double-click **HTML to XSLT**. Stylus Studio displays the **Open** dialog box.


4. Open `movies.html`, which is in the `examples\VideoCenter` directory of your Stylus Studio installation directory.

Stylus Studio displays a stylesheet in the XSLT Editor. The stylesheet contains one template, which matches the root node (`<xsl:template match="/">`). Stylus Studio copies all HTML markup that is in `movies.html` into the CDATA section of this template.

If you want, you can move and enlarge the XSLT editor window. If you do, leave a few inches at the bottom of the Stylus Studio window for the **Preview** window, which Stylus Studio will display later in this procedure.

Tip

An alternative to the previous four steps is to display `movies.html` in Internet Explorer and then select **Open with Stylus Studio**.

5. In the XSLT editor tool bar, click **Preview Result** .

Stylus Studio displays the **Scenario Properties** dialog box. A scenario associates a stylesheet with an XML source document. When Stylus Studio imports an HTML file, it automatically creates the stylesheet that can generate that HTML file.

6. In the **Scenario Properties** dialog box, in the **Scenario Name:** field, type `Video Scenario 1`.

7. Next to the **Source XML Url:** field, click **Browse** . Stylus Studio displays the **Open** dialog box.

8. In the **Open** dialog box, double-click `videos.xml`, which is in the `examples\VideoCenter` directory of your Stylus Studio installation directory.

9. In the **Base URL for HTML Links Resolution:** field, type the name of the directory in which Stylus Studio is installed followed by `\examples\VideoCenter`:
`Stylus_Studio_install_dir\examples\VideoCenter`

You must type an absolute path.

10. Click **OK**.

Stylus Studio displays the **Save As** dialog box.

11. In the **URL** field, type `video.xsl` and click the **Save** button.

Stylus Studio applies the new stylesheet to the `videos.xml` document and displays the result in the **Preview** window.

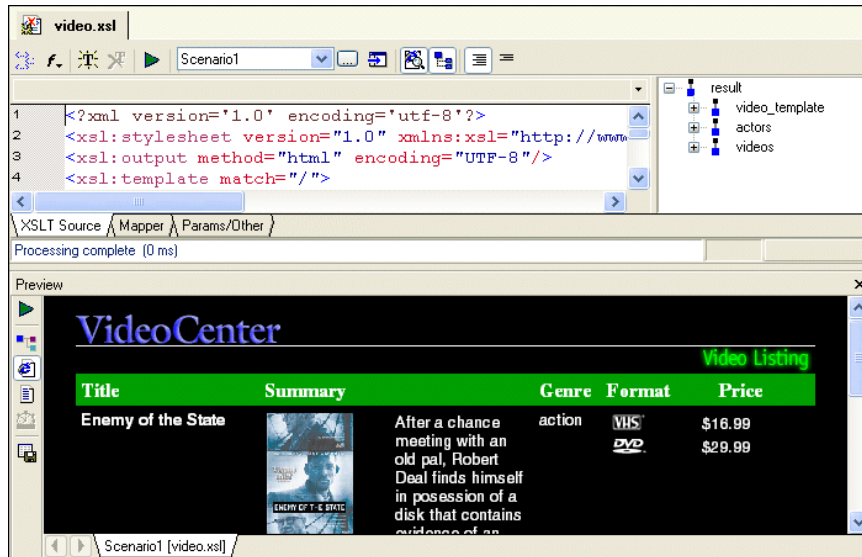


Figure 29. XSLT Preview of video.xsl

The result provides information about one video. Also, Stylus Studio displays the tree of the XML source document in a pane to the right of the stylesheet.


What to do next

Modify the stylesheet in the **XSLT Source** tab. Continue to the next topic, [“Creating the video Template”](#) on page 41.

Creating the video Template

◆ **To create the video template, follow these steps:**

1. In the source tree pane of the XSLT editor, click the plus sign next to the **videos** element.

Tip If the source tree pane is not visible, click **Source Tree**  at the top of the XSLT window, or select **XSLT > Display Source Tree Pane**.

2. Double-click the **video** repeating element.

Stylus Studio creates a template that matches the `video` element. The new template is empty, and it appears in the XSLT Editor pane near the end of the stylesheet (around line 159). In the source tree pane, a check appears next to **video** to indicate that there is a template that matches **video**.

3. In the **Preview** window, click somewhere in the movie title *Enemy of the State*.

Stylus Studio displays the template it created that matches the root element, and flags the line that generates the title. This shows you about where the HTML markup starts

for a single video. In the next few steps, you are going to move the markup for the video element into its own template.

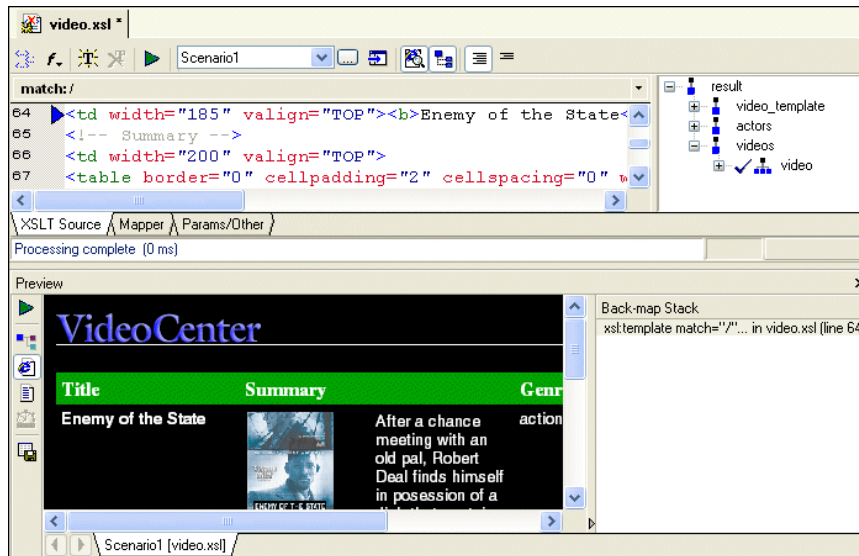


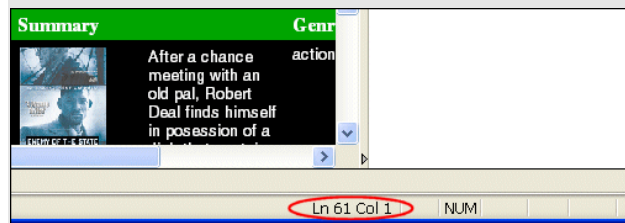
Figure 30. Back-mapping Shows What Source Generates Output

Tip When you backmap (by clicking in the output), Stylus Studio displays the **Back-map Stack** pane in the **Preview** window. This pane is discussed later in this topic.

4. In the XSLT Editor, highlight all markup that displays a single video.
 - a. Place the text cursor on line 61 – between the </TR> tag and the comment tag in the following line:


```
<!-- *** Start Video *** -->
```

Tip The cursor position (line and column number) is displayed in the Stylus Studio status bar, shown here:




- b. Drag your cursor down through the end of the following line, which is around line 110:

```
<!-- *** End Video *** -->
```

5. Press Ctrl+X to cut the highlighted text.
6. Scroll down to the video template.
7. Place your cursor on the blank line in the video template body (line 112) and press Ctrl+V to paste the video markup into its new template.
8. Click **Preview Result**  to apply the current stylesheet to the XML source document and refresh the **Preview** window.

The display no longer includes any information about the video. This is because you moved the video markup to the video template, but you did not specify an instruction that instructs the XSLT processor to instantiate the video template.

9. Click **Save** .

Instantiating the video Template

◆ **To instantiate the video template, follow these steps:**

1. In the XSLT Editor, put the text cursor in line 61 (the spot from which you cut the markup for the video element).

2. Type <x.

Stylus Studio displays the Sense:X tag completion menu.

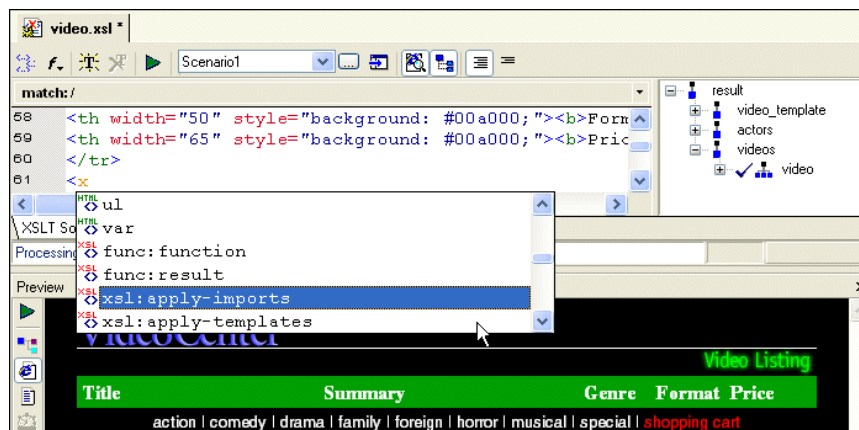


Figure 31. Sense:X Tag Completion in XSLT

3. Scroll down and double-click `xsl:apply-templates`.
Stylus Studio inserts the **xsl:apply-templates** instruction.
4. Type a space.
Stylus Studio again displays the choices for what you can enter.
5. Double-click **select**. A new SenseX: tag completion menu appears.
6. Double-click **result**.
7. Type a forward slash (/).
8. Double-click **videos** and type a forward slash (/).
9. Double-click **video** and type `</>`.

The XSLT you have just composed selects all video elements for processing. The complete instruction is as follows:

```
<xsl:apply-templates select="result/videos/video"/>
```

10. Click **Preview Result**  to refresh the **Preview** window.

Now you can see that the XSLT processor does instantiate the video template for each video element in the XML source document. However, because the template currently contains information for *Enemy of the State* only, the information for that video appears for each video element in the XML source document. You need to modify the XSLT instructions that generate the *Enemy of the State* text. Each time the XSLT processor instantiates the video template, you want it to pull in information for a different video.

11. Click **Save** .

Making Titles Dynamic

◆ To change the video template to display the title for each video, follow these steps:

1. In the **Preview** window, click somewhere in the movie title *Enemy of the State*.
In the **Backmap Stack** pane, Stylus Studio shows you the XSLT instruction that generated the output you clicked in the result document. In the XSLT Editor pane, Stylus Studio's back-mapping feature points to the line that contains *Enemy of the State*.
2. Delete the text *Enemy of the State*.
Do not cut the `` and `` tags.



3. In the spot in the line where you deleted `Enemy of the State`, type the following instruction, or select its parts from the Sense:X tag completion lists.

```
<xsl:apply-templates select="title"/>
```

This instructs the XSLT processor to instantiate a template that matches `title`. However, there is no such template because a `title` template is not needed in this stylesheet. The default templates provide the required operations. When the XSLT processor cannot find a template that matches `title`, it instantiates the default template that matches all elements — its template rule is `/|*`.

This default template operates on the children of the node for which it is instantiated. In this case, each time the default template is instantiated for a `title` element, it operates on the text node that is the lone child of the `title` element. Because the child node is a text node, this template instantiates the other default template, the one that matches all text nodes. This other default template copies the contents of the text node to the result document. This is how the actual title of each video gets displayed in the result.

The templates that match `/|*` and `text()|@*` are default templates in every stylesheet whether or not you or Stylus Studio explicitly include them. For additional information about how these templates work, see “[Understanding How the Default Templates Work](#)” on page 353.

4. Click **Preview Result**  to refresh the **Preview** window.
Now the display for each video has a different title, but the rest of the information (the summary, the image, and so on) is that for *Enemy of the State*.
5. Click **Save** .

Making Images Dynamic

- ◆ **To change the video template to display the image associated with each video, follow these steps:**



1. In the **Preview** window, click in the *Enemy of the State* image.
Again, Stylus Studio’s back-mapping feature identifies the line in the stylesheet that generates the image.
The **Backmap Stack** window now displays two `xsl:template` instructions. One matches the root node and one matches `video`.
2. To edit the expression that identifies the image, in the XSLT editor pane, delete `id1244100` from the line Stylus Studio is pointing to.

3. Replace the specific ID with the following attribute value template:
{@id}

The result should look like the following:

```

```

Again, the XSLT processor will use the default templates to copy the text to the result document.
4. Click **Preview Result**  to refresh the **Preview** window.
Stylus Studio displays a different summary for each title.
5. You can follow the same procedure described here to make the other child elements of the video element (genre, rating, and so on) dynamic. But before you do that, save the work you have already done.
6. Click **Save** .

Using the XSLT Mapper – Getting Started



The XSLT Mapper is available only in Stylus Studio XML Enterprise Suite and Stylus Studio XML Professional Suite.

This section helps you get started using the XSLT Mapper to create stylesheets that aggregate data and transform XML. The sample files used in this section are in the Stylus Studio `examples\simpleMappings` directory. If you follow the procedures in this section, you create the `BooksToCatalog.xsl` stylesheet. A sample version of this stylesheet, `sampleBooksToCatalog.xsl`, is also in the `examples\simpleMappings` directory of your Stylus Studio installation directory.

Each of the topics in this section contains instructions for working with sample XML documents that you can use to familiarize yourself with the XSLT Mapper. You should perform the steps in each topic before you move on to the next topic – after the first topic, some steps depend on actions you performed in a previous topic.

This section covers the following topics:

- [“Opening the XSLT Mapper”](#) on page 47
- [“Mapping Nodes in Sample Files”](#) on page 50
- [“Saving the Stylesheet and Previewing the Result”](#) on page 53
- [“Deleting Links in Sample Files”](#) on page 54
- [“Defining Additional Processing in Sample Files”](#) on page 54

In addition to the topics described in this section, the *Stylus Studio® 2007 User Guide* contains other sources of information on XSLT:

- To learn more about XSLT, see [“Working with XSLT”](#) on page 325.
- To get started XSLT Editor features for stylesheets, see [“Working with Stylesheets – Getting Started”](#) on page 27.
- To learn about the XSLT mapper in greater detail, see [“Creating XSLT Using the XSLT Mapper”](#) on page 449.

To get started, you will need to start Stylus Studio if you haven’t already. See [“Starting Stylus Studio”](#) on page 5 if you need help with this step.

Opening the XSLT Mapper

This procedure describes how to open the XSLT Mapper and select the files you want to use for the drag-and-drop operations that will define your XSLT stylesheet.

◆ **To open the XSLT Mapper:**

1. From the Stylus Studio menu bar, select **File > New > XSLT Stylesheet**.
Stylus Studio displays the **Scenario Properties** dialog box.
2. Click the **Cancel** button to dismiss the dialog box.
3. Click the **Mapper** tab.

Stylus Studio displays XSLT editor with the **Mapper** tab selected. The source pane beneath the mapper panes appears by default, allowing you to see how the mappings of XML document elements are rendered as XSLT. The source pane is fully editable and synchronized with the XSLT Mapper. Of course, you can always click the **XSLT Source** tab for a full-screen view of your XSLT code.

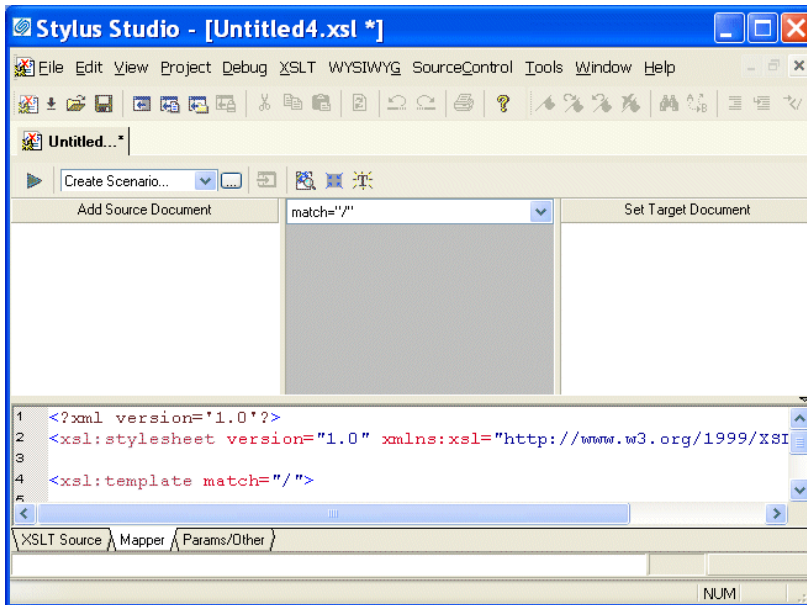


Figure 32. XSLT Editor Mapper Tab for a New Stylesheet

Tip

The **Project** window also appears if it was open the last time Stylus Studio was closed. You can close it.

4. Click the **Add Source Document** button at the top of the mapper's left pane.
Stylus Studio displays the **Open** dialog box.

5. For this example, navigate to the `examples\simpleMappings` directory in the Stylus Studio installation directory.
6. Double-click **books.xml**.
7. Click the **Set Target Document** button at the top of the Mapper's right pane. Stylus Studio displays the **Open** dialog box.
8. For this example, navigate to the `examples\simpleMappings` directory in the Stylus Studio installation directory.
9. Double-click **catalog.xml**.

Stylus Studio displays tree diagrams of these XML documents. The default XSLT source code has not been altered at this point.

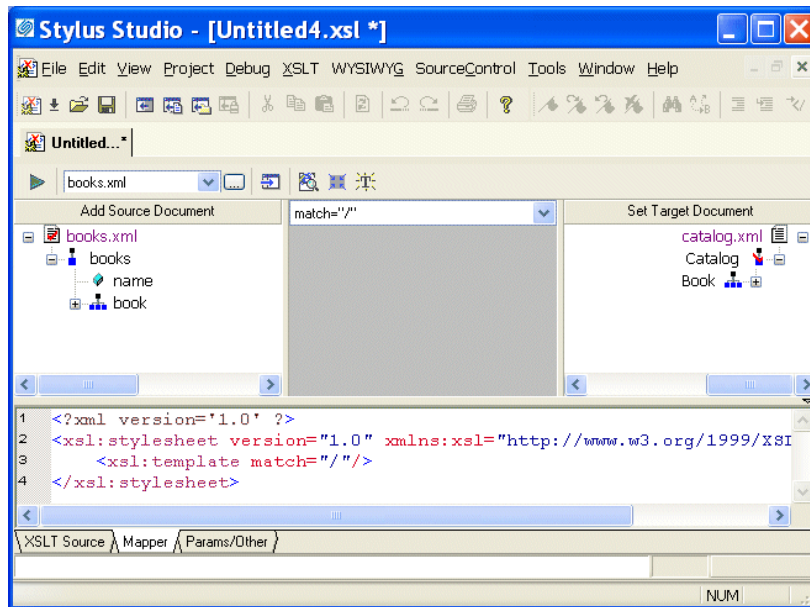


Figure 33. XSLT Mapper Tab with Source and Target Documents

Mapping Nodes in Sample Files

◆ **To define links and examine the stylesheet Stylus Studio creates:**

1. In the **Mapper** tab, expand the tree for both `books.xml` and `catalog.xml`.

Tip You can display an entire tree using the asterisk key (*) on your keyboard's number pad.

2. In `books.xml`, place the pointer over the book repeating element.
3. Press and hold the left mouse button, and drag from `book` to the `Book` repeating element in `catalog.xml`.

Stylus Studio draws a line as you drag.

4. Release the mouse button to create the link between `book` and `Book`.

Stylus Studio creates an `xs1:for-each` block that links the `book` and `Book` repeating elements. (If you mouse over the block, `xs1:for-each` appears in a pop-up to indicate the XSLT operation represented by the link.)

Tip If you prefer, you can render `xs1:for-each` as a simple line. You might want to do this to simplify the appearance of the mapper canvas. Select **Tools > Options** from the menu, and then navigate to **Module Settings > XSLT Editor > Mapper**.

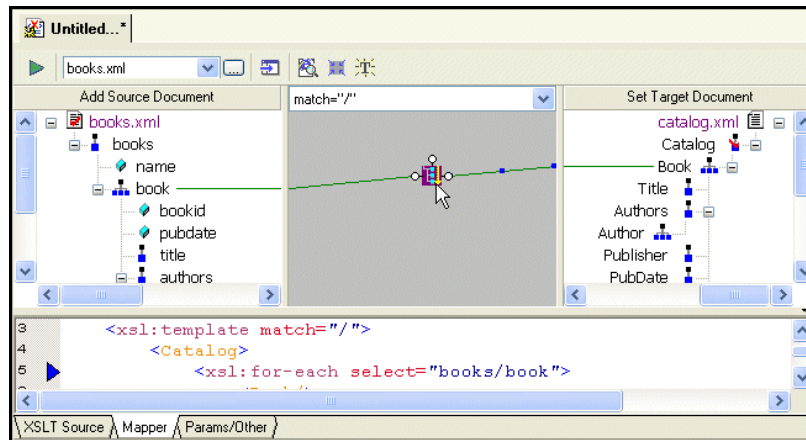


Figure 34. xs1:for-each Block Displayed by the XSLT Mapper

Also notice that the complete `xs1:for-each` instruction has been added to the XSLT source, which appears in the XSLT source pane under the XSLT Mapper canvas. The back-mapping pointer identifies the line of XSLT that was just added to stylesheet.

The template contains an `xs1:for-each` instruction that selects the `book` element, which is the node you selected in [Step 2](#). The output from this template is an empty `Book` element, which is the node that was the target of the link. Stylus Studio created the `Catalog` element automatically, to provide the document structure necessary to support the `Book` element.

Tip By default, Stylus Studio creates an `xs1:value-of` instruction when you link one element to another; Stylus Studio creates an `xs1:for-each` instruction if you link two repeating elements. You can also create other types of instructions graphically, including `xs1:if`, `xs1:choose`, and `xs1:apply-template`.

5. Click the **Params/Other** tab.

In the **Output method:** field, display the drop-down list and click **html**. Note, however, that the output of a stylesheet generated by the XSLT Mapper is always XML – even if the setting for **Output method** is **unspecified**, Stylus Studio still generates XML.

6. Click the **Mapper** tab.

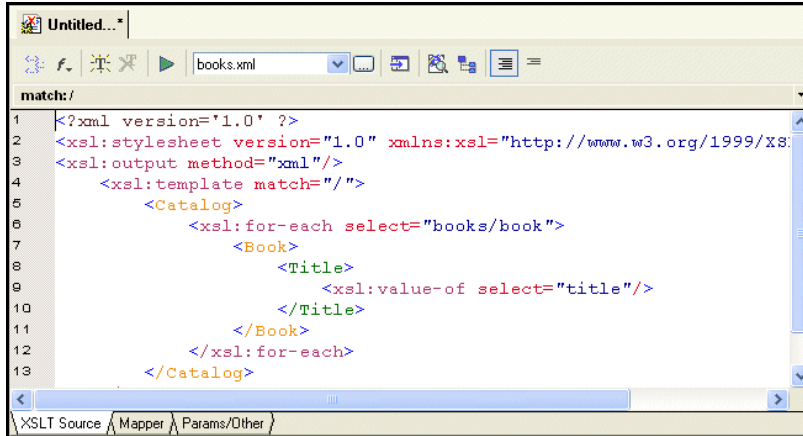
The `xs1:output` instruction is added to the XSLT source:

```
<xs1:output method="html"/>
```

7. Create another link from the **title** element to the **Title** element.

Note When you map, you always map from the source document to the destination document.

- Click the **XSLT Source** tab to see the new instructions in the template. (If you prefer, you can simply adjust the splitter between the XSLT source pane and the XSLT Mapper canvas.)

The screenshot shows the Stylus Studio interface with a file named 'books.xml' open. The main editor displays XSLT code for a catalog. The code starts with an XML declaration and an XSLT stylesheet declaration. It then defines a template that matches the root node. Inside the template, there is a <xsl:for-each> loop that iterates over 'books/book' elements. For each iteration, it creates a <Book> element. Inside the <Book> element, it creates a <Title> element. The <Title> element's content is determined by the <xsl:value-of> instruction, which selects the 'title' child of the current context node. The code is as follows:

```
1 <?xml version='1.0' ?>
2 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL
3 <xsl:output method="xml"/>
4 <xsl:template match="/">
5   <Catalog>
6     <xsl:for-each select="books/book">
7       <Book>
8         <Title>
9           <xsl:value-of select="title"/>
10        </Title>
11      </Book>
12    </xsl:for-each>
13  </Catalog>
```


Figure 35. Stylus Studio Builds XSLT Based on the Mapper Links

For each link you define, Stylus Studio adds instructions to the template that matches the root node. In the XSLT you have composed so far, the XSLT inserts a `Book` element for each `book` element it finds in the source document. In the `Book` element, the stylesheet selects the `title` elements. For each `title` element, it inserts a `Title` element. Finally, in each `Title` element, the stylesheet extracts the value of the current context node, which is the `title` node.

Why does the stylesheet extract the value of the `title` nodes but not the `book` nodes? The `title` node has only a text node as its child. In this situation, the default is that the XSLT Mapper inserts an `xsl:value-of` instruction.

Saving the Stylesheet and Previewing the Result

◆ To save the stylesheet and preview the result:

1. Click **Save** . Stylus Studio displays the **Save As** dialog box.
2. In the **URL:** field, type BooksToCatalog.xml.
3. Click the **Save** button.

This saves the stylesheet that Stylus Studio has generated. It does not matter that you have not finished mapping all nodes.

4. In the upper left corner of the XSLT Mapper, click **Preview Result** .

Tip


When you create a stylesheet using the XSLT Mapper, Stylus Studio automatically creates a scenario for you, using the source document you specify as the source document for the scenario. Scenarios and their value in the application development process are described earlier in this chapter. See “XSLT Scenarios” on page 32.

Stylus Studio displays the result of processing books.xml with the stylesheet you created in the XSLT Mapper in the **Preview** window.



Figure 36. Result of Applying XSLT to books.xml

The result document uses the same schema as the target document, catalog.xml in this example. Because not all nodes have been mapped yet, the result document does not contain all nodes found in books.xml (author and subject nodes, for example).

5. You can confirm that the result document is incomplete by viewing books.xml. Click **Open XML From Scenario** , which is at the top of the **Mapper** tab. Stylus Studio displays the books.xml document in the Stylus Studio XML Editor.
6. Review the XML document, and then click the document tab for the **BooksToCatalog.xml** stylesheet to re-display the XSLT Editor.

Deleting Links in Sample Files

◆ To delete links:

1. Click the **Mapper** tab if it is not already selected.
2. Click the `title` to `Title` link to select it.

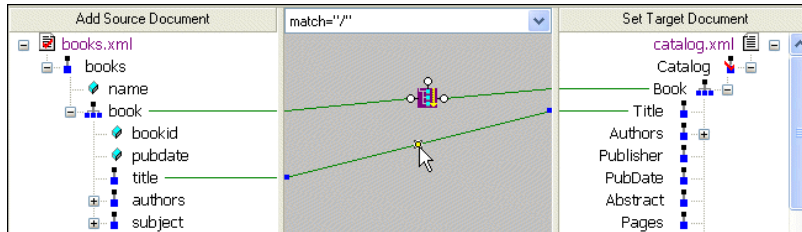


Figure 37. Click a Link to Select It

3. Press the Delete key, or
 - a. Right-click the selected link. This displays a shortcut menu.
 - b. Click **Delete** to delete the selected link.

Tip

In addition to **Delete**, the shortcut menu displays the following options:

- **Go To Source** displays the line of XSLT code represented by the link you select in the XML Editor.
- **Carry Value** allows you to create `<xsl:value-of select="."/>` statements. This option is available for links representing `xsl:for-each` instructions only.

Defining Additional Processing in Sample Files

The stylesheet that the XSLT Mapper creates is not limited to the instructions that Stylus Studio adds. You can edit the template as you would any template. Stylus Studio automatically incorporates any changes you make to the template and displays them in the **Mapper** tab, if it is appropriate to do so.

In addition, you can perform external processing by, for example, defining Java functions and incorporating those functions in your XSLT stylesheet. Like standard supported XSLT functions, user-defined Java functions can be created graphically in the XSLT Mapper – just right click on the mapper canvas, select **Java Functions** from the shortcut menu, and select any registered Java function you want to use.

See “[Processing Source Nodes](#)” on page 476.

Debugging Stylesheets – Getting Started

The Stylus Studio debugger allows you to follow XSLT processing and detect errors in your stylesheets. Stylus Studio includes sample files that you can experiment with to learn how to use the debugger. To get you started, this section provides step-by-step instructions for using the debugger with these sample files. You should perform the steps in each topic in the order of the topics.

For complete information about how to use the debugger, see [“Debugging Stylesheets”](#) on page 493.

In addition, Stylus Studio allows you to observe and debug the interaction between your Java code and XML data. See [“Debugging Java Files”](#) on page 503.

This section includes the following topics:

- [“Setting Up Stylus Studio to Debug Sample Files”](#) on page 55
- [“Inserting a Breakpoint in the Sample Stylesheet”](#) on page 56
- [“Gathering Debug Information About the Sample Files”](#) on page 58

To get started, you’ll need to start Stylus Studio if you haven’t already. See [“Starting Stylus Studio”](#) on page 5.


Setting Up Stylus Studio to Debug Sample Files

◆ **To set up Stylus Studio to debug sample files:**

1. Open the `videosDebug.xsl` stylesheet, located in the `examples\VideoCenter` directory where Stylus Studio was installed.

Alternative: If the Stylus Studio **Project** window is open, you can access this stylesheet from the `examples` project.

Stylus Studio displays the `videosDebug.xsl` stylesheet in the XSLT editor.

2. In the XSLT editor tool bar, click **Preview Result**  to run the predefined scenario `DebugVideosScenario`. The source XML document is `videos.xml`.

Stylus Studio applies the stylesheet and displays the results (a finished HTML page that displays information about a single video) in the **Preview** window.

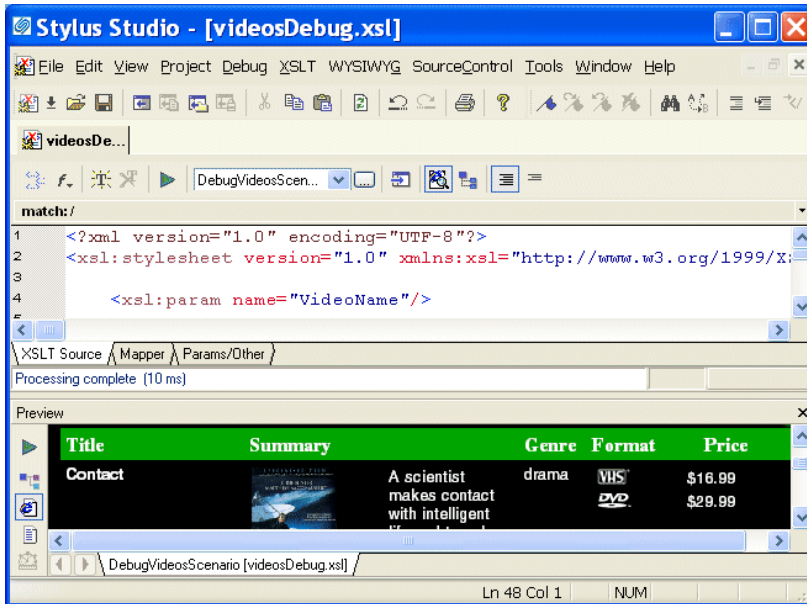


Figure 38. Preview of videosDebug.xml

Inserting a Breakpoint in the Sample Stylesheet

As with any debugger, in the Stylus Studio XSLT debugger you insert a breakpoint where you want to suspend processing and examine what is going on. You can do this using the **Debug** menu or the debug set of tools in the tool bar.

Tip Tools in the tool bar are in grouped by function. These groups, like the one for debug tools shown here, are dockable and can be moved anywhere you please.



◆ **To insert a breakpoint in the sample stylesheet:**

1. In the XSLT Editor, click in line 202. Line numbers appear in the lower right corner of the XSLT Editor window. Line 202 starts with `<xsl:template match="director">`

Tip To display lines in Stylus Studio text editors, click **Tool > Option > Editor General**, and select **Show line numbers**.

2. In the Stylus Studio tool bar, click **Toggle Breakpoint**  .

Alternative: If you prefer, select **Debug > Toggle Breakpoint**, or press F9.

Stylus Studio displays a red circle to the left of the line that contains the `xsl:template match="director"` instruction. The XSLT processor will stop processing when it gets to the instantiation of this template.

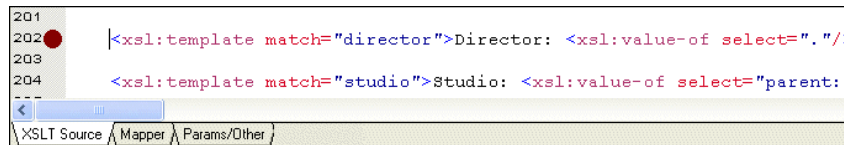



Figure 39. A Red Circle Shows Where Breakpoints Are Set

Do not do it, but to remove a breakpoint, you click in the line that has the breakpoint and then click **Toggle Breakpoint** (or F9). The **Toggle Breakpoint** button and F9 key operate as toggles.

3. Press F5 to start debugging.

Alternative: In the Stylus Studio tool bar, click **Start Debugging**  .

The XSLT processor displays a yellow triangle to indicate where processing has been suspended. Instead of the finished HTML created when you first ran the scenario, the **Preview** window displays just the HTML code because complete processing of the XSLT was suspended before the finished HTML could be rendered.

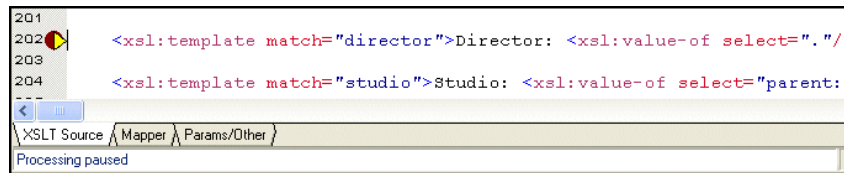




Figure 40. Yellow Triangle Shows Where XSLT Processing Stopped

Do not do it, but to stop debugging, you can click **Cancel** in the lower right corner of the XSLT editor window, or click **Stop Debugging**  in the Stylus Studio tool bar.

If you click **Preview Result**  instead of pressing F5, Stylus Studio applies the stylesheet without running the debugger. Pressing F5 always invokes the debugger. If there are no breakpoints, and no errors, processing completes and Stylus Studio displays the result in the **Preview** window.

Gathering Debug Information About the Sample Files

When XSLT processing is suspended at a breakpoint, Stylus Studio displays the **Variables**, **Call Stack**, and **Watch** windows.

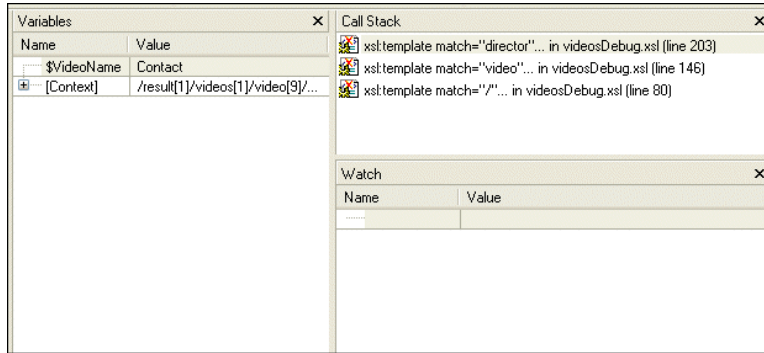
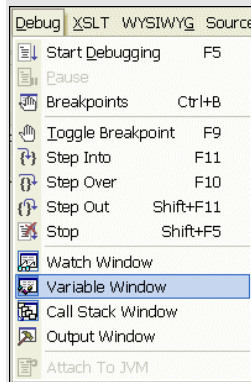


Figure 41. Variable, Call Stack, and Watch Windows Appear During Debugging

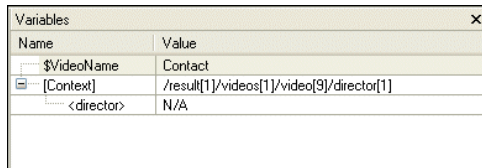
You can use the information in these windows to learn about potential and actual problems encountered in your XSLT processing.

Tip You can also control the display of these windows using the **Debug** menu, shown here, or the tool bar.



The Variables Window

The **Variables** window displays a list of variables and their values when processing was suspended.



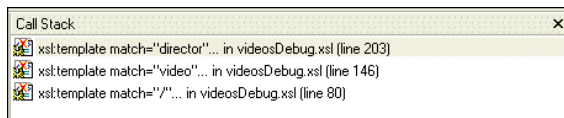
Name	Value
\$VideoName	Contact
[Context]	/result[1]/videos[1]/video[9]/director[1]
<director>	N/A

Figure 42. Variables Window

As you can see, the stylesheet defines the `VideoName` parameter, which had no value when processing was suspended. In addition, the **Variables** window shows you that when processing was suspended, the processor was operating on the first `director` child element of the first `video` child element of the first `videos` child element of the first `result` element.

The Call Stack Window

The **Call Stack** window displays a history of the steps the processor performed to reach the point at which processing was suspended, including the names of the templates that are currently instantiated, in most recent-to-oldest order.



xsl:template match="director"... in videosDebug.xml (line 203)
xsl:template match="video"... in videosDebug.xml (line 146)
xsl:template match="/"... in videosDebug.xml (line 80)

Figure 43. Call Stack Window

In this example, the XSLT processor has instantiated the `director` template, which is part of the instantiation of the `video` template, which is part of the instantiation of the template that matches the root node.

- ◆ **To step out of debug** Step out , or press **Shift+F11**.

The processor completes the instantiation of the director template, which adds some HTML to the **Preview** window. The yellow triangle moves to show the new location in the XSLT source.

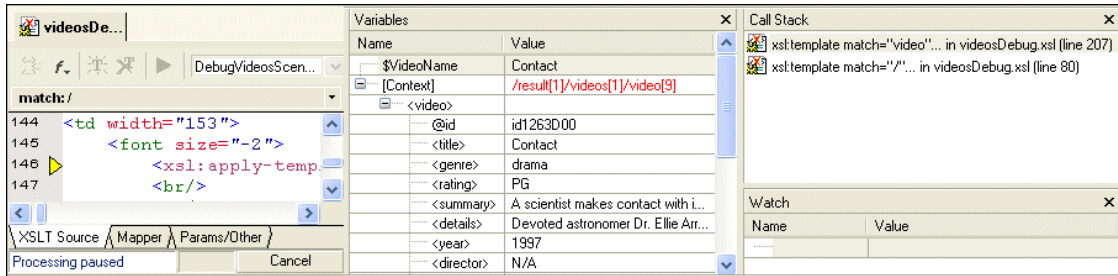


Figure 44. Stepping Out Advances the Processor

As you can see in the **Call Stack** window, the processor is now two levels deep in the template that matches the root node, instead of three levels deep as it was previously. The value of the context node in the **Variables** window is `/result[1]/videos[1]/video[9]` (it was `/result[1]/videos[1]/video[9]/director[1]`).

The Watch Window

If your application contains a lot of variables, the **Watch** window allows you to focus on the variables in which you are particularly interested.

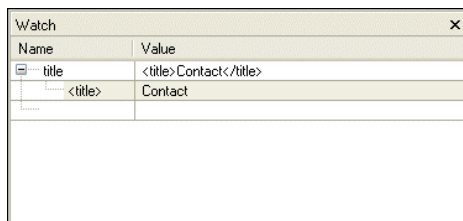


Figure 45. Watch Window Lets You Track Variables

- ◆ **To enter a variable to watch:**

1. Double-click the **Name** field.
2. Type the name of the variable you want to watch and press Enter.

As processing continues, the **Watch** window displays the values of the variables you specify.

Ending Processing During a Debug Session

◆ To end processing during a debug session:

1. In the **Preview** window, click any line of text.
In the **XSLT Source** tab, Stylus Studio displays the blue back-mapping triangle that indicates the line in the stylesheet that generated the output line you clicked.
2. In the lower right corner of the XSLT editor, click the **Cancel** button to end processing.

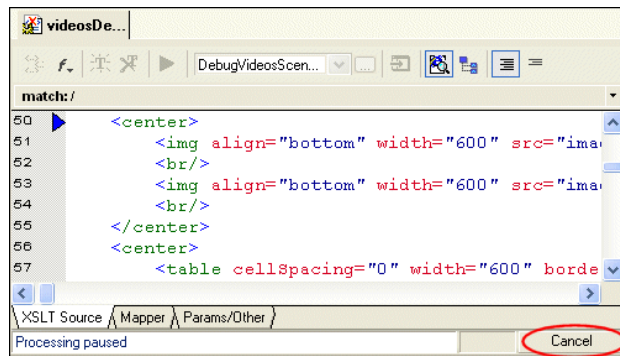


Figure 46. Cancelling Processing During Debugging

Stylus Studio displays a notification message that indicates that processing has been stopped and, optionally, allows you to jump to the location where processing ended.

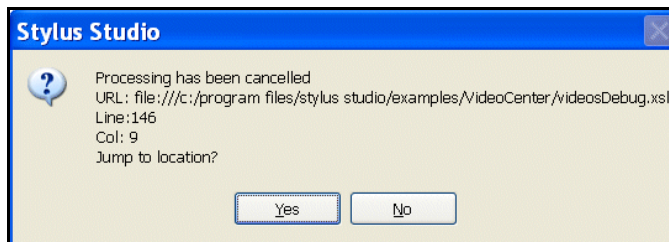



Figure 47. You Can Jump to Where XSLT Processing Ended

3. Click **Yes** to jump to the location where processing ended.
The cursor appears on line 146 of the **XSLT Source** tab, which contains `<xsl:apply-templates select="director"/>`.

4. In the XSLT editor tool bar, click **Open XML from Scenario** . Stylus Studio displays the XML source document that the stylesheet operates on. As you can see, the first result element is the document element.

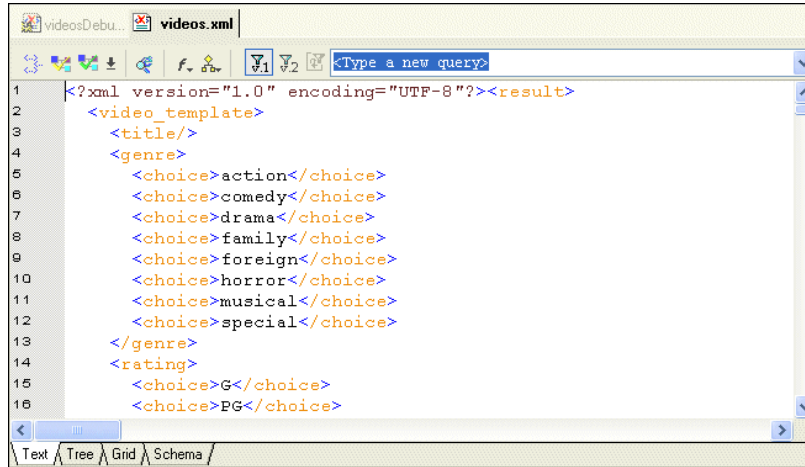


Figure 48. Viewing the XML Source for an XSLT Transformation

This section demonstrated some of the major features of Stylus Studio's debug tools, including specialized windows for presenting call stack and variable information. For complete information on using the XSLT debugger, see [“Debugging Stylesheets”](#) on page 493.

Defining a DTD – Getting Started

This section provides a quick tour of the main features of the DTD Editor. It provides instructions that you can follow to actually define a simple DTD. For complete documentation about how to use the Stylus Studio DTD Editor, see [“Defining Document Type Definitions”](#) on page 603.

Process Overview

When you use Stylus Studio to define a DTD, the main steps you perform are:

1. Create a new DTD schema file.
2. Define the elements that contain the raw data.
3. Define the elements that contain other elements.
4. In the container elements, specify the rules for the contained elements. That is, specify whether a contained element is optional or required, whether there can be more than one, and what order contained elements must be in.


This section provides step-by-step instructions for defining the bookstore.dtd schema file. You should perform the steps in each topic in the order of the topics. This section includes the following topics:

- [“Creating a Sample DTD”](#) on page 63
- [“Defining Data Elements in a Sample DTD”](#) on page 64
- [“Defining the Container Element in a Sample DTD”](#) on page 65
- [“Defining Structure Rules in a Sample DTD”](#) on page 65
- [“Examining the Tree of a Sample DTD”](#) on page 67

To get started, you’ll need to start Stylus Studio if you haven’t already. See [“Starting Stylus Studio”](#) on page 5.

Creating a Sample DTD

◆ **To create a new DTD schema file:**

1. From the Stylus Studio menu bar, select **File > New > DTD Schema**.
2. Click **Save** .


Stylus Studio displays the **Save As** dialog box.

3. Navigate to the Stylus Studio examples directory.
4. In the **URL:** field, type bookstore.dtd.
5. Click the **Save** button.

Defining Data Elements in a Sample DTD

In your DTD, suppose you want a book element to be optional. Further, if a book element is present, it must always have exactly one title element and it can have any number of author elements. The title and author elements contain only raw data.

◆ **To accomplish this, perform the following steps:**

1. At the bottom of the DTD editor, click the **Tree** tab.
2. Click the **DTD** node at the top of the tree if it is not already selected.
3. Click **New Element Definition** , which is the top button in the tool bar on the left side of the DTD editor window.
Stylus Studio displays an entry field for the element name.
4. Type title and press Enter.
Stylus Studio displays the new element, title, and the element's properties in the **Properties** window.

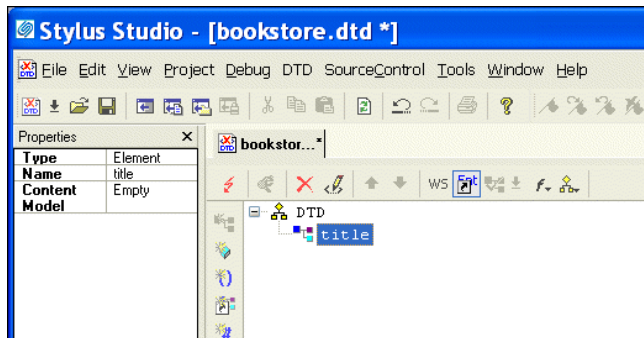



Figure 49. New Element in the DTD Editor

5. Click **New Modifier** .
- Stylus Studio displays an entry field for the element's modifier.
6. Double-click **Zero or More**.
The new modifier is added to the element.

7. Click **Add #PCDATA** .
8. To define the author element, repeat [Step 2](#) through [Step 7](#). In [Step 4](#), type author instead of title.

When you are done, the Stylus Studio desktop should resemble the following:

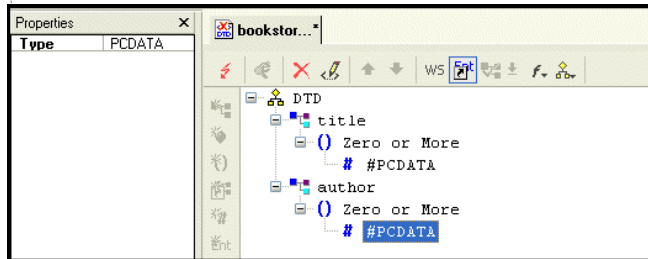





Figure 50. Creating a DTD with Two Elements



Defining the Container Element in a Sample DTD

- ◆ **To define the book element:**
 1. Click the **DTD** node at the top of the tree.
 2. Click **New Element Definition** .
 3. Type book and press Enter.

Defining Structure Rules in a Sample DTD

- ◆ **To specify the rules for the structure of the book element:**
 1. Click the **book** node in the DTD tree if it is not already selected.
 2. Click **New Modifier** .
 3. In the drop-down list that appears, scroll down and double-click **Sequence**. This indicates that the book element can include one or more elements.
 4. Click **New Reference to Element** .
 5. Type title in the entry field and press Enter.

Because the reference to the title element appears immediately after the Sequence modifier, the DTD editor assumes that the default behavior is what is wanted. That is, the book element must contain exactly one instance of the title element.

6. Click the **Sequence** modifier.
7. Click **New Modifier** .
8. Double-click **One or More**. (There can be one or more author elements in each book element.)
9. Click **New Reference to Element** .
10. Type author in the entry field and press Enter.

At this point, the definition of the book element is complete, and the tree diagram of bookstore.dtd should look like this:

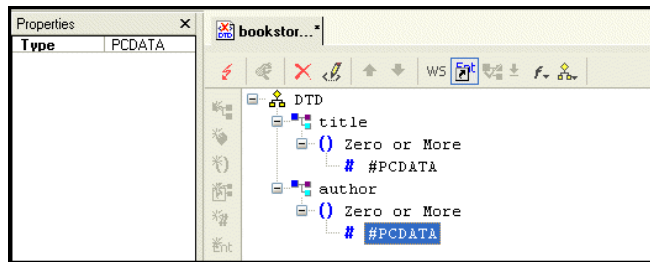






Figure 51. Early Steps of bookstore.dtd

However, you have not yet specified that you want the book element itself to be optional. You need to do this in the element that references the book element. For example, suppose the bookstore element is the root element in XML documents that use this DTD. Further suppose that you want the book element to be a child of the bookstore element.

◆ **You can define the bookstore element as follows:**

1. Click the **DTD** node at the top of the tree.
2. Click **New Element Definition** .
3. Type bookstore in the entry field and press Enter.
4. Click **New Modifier** .
5. In the drop-down list that Stylus Studio displays, double-click **Optional**.
6. Click **New Reference to Element** .
7. Type book in the entry field and press Enter.
8. Click **Save** .

Examining the Tree of a Sample DTD

Your DTD should now look like [Figure 52](#).

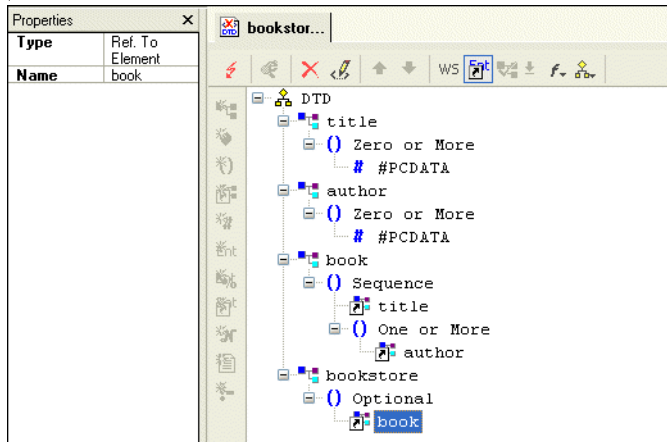


Figure 52. Finished bookstore.dtd

To complete this DTD, you could define magazine and newsletter elements. In the bookstore element definition, you could add references to the magazine and newsletter elements. You could also expand the definition of the book element to include information about the publisher, price, publication date, and number of pages.

Defining an XML Schema Using the Diagram Tab – Getting Started

This section provides a quick tour of the main features of the **Diagram** tab of the XML Schema Editor and shows you how to define a simple XML Schema. For complete documentation about how to use the XML Schema Editor, see [“Creating an XML Schema in Stylus Studio”](#) on page 510.

The topics in this section provide step-by-step instructions for defining the bookstoreDiagram.xsd XML Schema document. You should perform the steps in each topic in the order of the topics.

This section includes the following topics:

- [“Introduction to the Diagram Tab”](#) on page 69
- [“Editing Tools of the Diagram Tab”](#) on page 78
- [“Description of Sample XML Schema”](#) on page 83
- [“Defining a complexType in a Sample XML Schema in the Diagram View”](#) on page 83
- [“Defining Elements of the Sample complexType in the Diagram View”](#) on page 91

To get started, you will need to start Stylus Studio if you have not already. See [“Starting Stylus Studio”](#) on page 5.

Introduction to the Diagram Tab

The recommended way to define an XML Schema in Stylus Studio is to start with the **Diagram** tab of the XML Schema Editor, which is shown in [Figure 53](#).

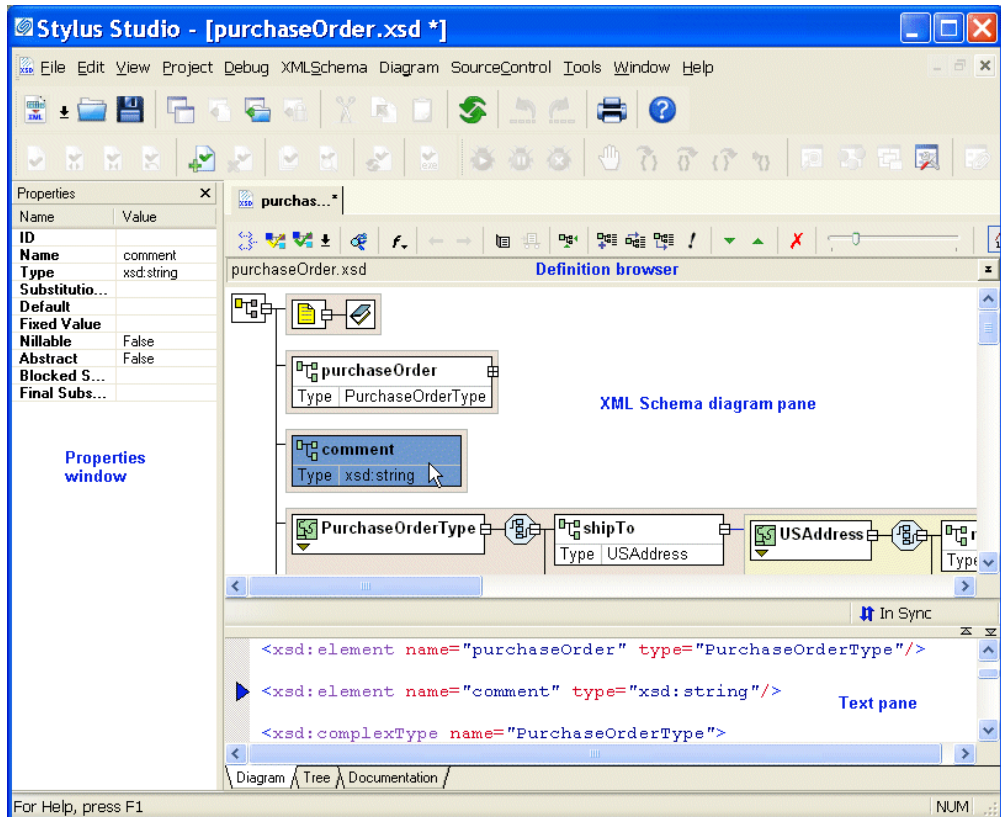


Figure 53. Diagram Tab of the XML Schema Editor

When you use the **Diagram** tab to define an XML Schema, you can create XML Schema nodes directly on the *XML Schema diagram pane* using tools on the tool bar or from the **XML Schema > Diagram** and shortcut menus. You can also type in the *text pane*, which appears under the **Diagram** tab. The text pane displays the XML Schema syntax Stylus Studio creates for you as you work in the diagram pane.

Stylus Studio ensures that the XML Schema you create is valid. For example, any nodes you define are created in the required order in the XML document that contains the XML Schema definition, regardless of the order in which you create them.

The **Diagram** tab, shown in [Figure 53](#), consists of three main areas:

- “[Diagram Pane](#)” on page 70
- “[Text Pane](#)” on page 75
- “[Definition Browser](#)” on page 77

This section describes these areas and how to work with them.

Diagram Pane

The diagram pane contains graphical representations of the elements, attributes, and other nodes that make up your XML Schema.

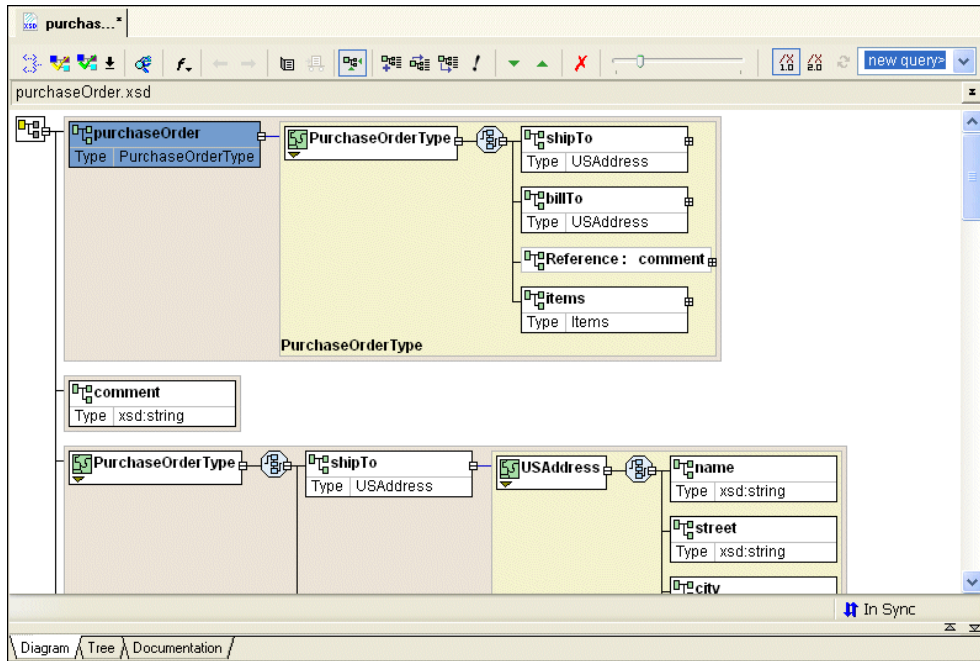






















Figure 54. Schema Diagram Pane

Nodes

Each node displayed in the diagram pane is represented by its own symbol; tool tips, which are displayed when you hover over a node in the diagram, identify the node’s type

(element, attribute, sequence, and so on). The symbols used in the diagram are summarized in [Table 1](#).

Table 1. Symbols Used in the XML Schema Diagram

<i>Symbol</i>	<i>Meaning</i>
	schema (xsd:schema)
	annotation (xsd:annotation)
	documentation (xsd:documentation)
	element (xsd:element)
	attribute (xsd:attribute)
	attributeGroup (xsd:attributeGroup)
	simpleType (xsd:simpleType)
	complexType (xsd:complexType)
	choice (xsd:choice)
	sequence (xsd:sequence)
	key (xsd:key)
	key reference (xsd:keyref)
	unique (xsd:unique)
	group (xsd:group)
	simpleContent (xsd:simpleContent)
	complexContent (xsd:complexContent)
	restriction (xsd:restriction)
	extension (xsd:extension)
	union (xsd:union)
	list (xsd:list)

Nodes can be expanded and collapsed using the plus and minus symbols, respectively, that appear on the right side of the node. In [Figure 57](#) for example, the PurchaseOrderType complexType has been expanded. The shipTo element has not.

Displaying Properties

To streamline the diagram, most nodes are displayed with their properties hidden by default. Exceptions include element, extension, and restriction nodes, for which the type is displayed, as shown in the **productName** element in [Figure 55](#).

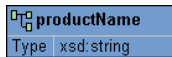


Figure 55. Most Nodes Appear with Properties Hidden

You can change the display for classes of nodes (all elements, for example) using the **Schema Diagram Properties** dialog box, shown in [Figure 56](#). (In addition, the **Properties** window displays all the properties for any node you select.)

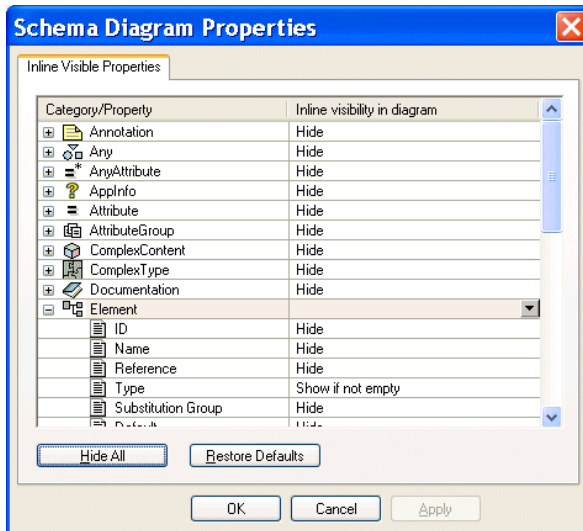


Figure 56. Schema Diagram Properties Dialog Box

For each node property, you can choose to

- Show the property
- Show the property only if it is not empty
- Hide the node

If all of a node's properties have the same show/hide setting, that value is displayed in the **Inline Visibility in Diagram** field. If no value is displayed in the **Inline Visibility in Diagram** field, it means that two or more properties have different show/hide settings.

◆ **To display the Schema Diagram Properties dialog box:**

- Select **Diagram > Properties** from the Stylus Studio menu
- Select **Properties** from the diagram shortcut menu

Tip You can also change schema diagram properties on the **Diagram** page of the **Options** dialog box – **Tools > Options > Module Settings > XML Schema Editor > Diagram**.

◆ **To change node properties display:**

1. Display the **Schema Diagram Properties** dialog box, or the **Diagram** page of the **Options** dialog box.
2. Select the node whose properties display you want to change.

Tip To hide all properties, click the **Hide All** button. To restore defaults, click the **Restore Defaults** button.

3. Click **OK**.

Background color

Background color is used as another visual cue for information about the XML Schema:

- A tan, or light brown, color identifies global nodes – these are elements, types, and so on, that are defined as children of the schema (xsd:schema). In [Figure 57](#), the purchaseOrder element is an example of a globally defined node.

- A light yellow background identifies local instances of globally defined types. In [Figure 57](#), the PurchaseOrderType complexType is a local instance of that type.

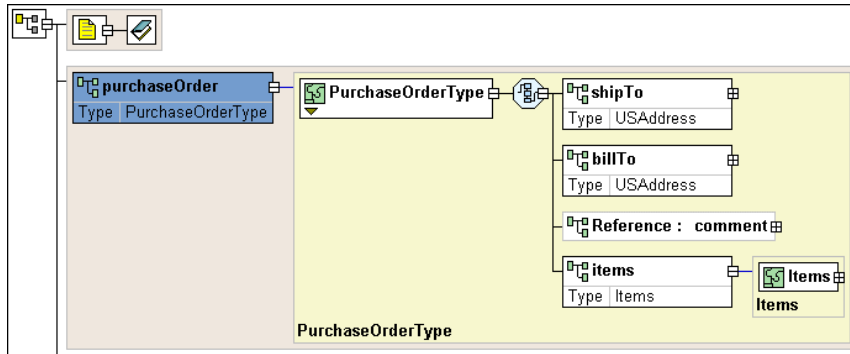


Figure 57. Background Colors Show Global and Local Types

Displaying documentation

By default, text associated with documentation elements (`xsd:documentation`) is hidden. You can expand documentation elements in the diagram by clicking the **Show Documentation** (📄) button, or by selecting **Diagram > Show Documentation** from the Stylus Studio menu. When you do, the text associated with all documentation elements defined in the XML Schema appears, as shown in [Figure 58](#).

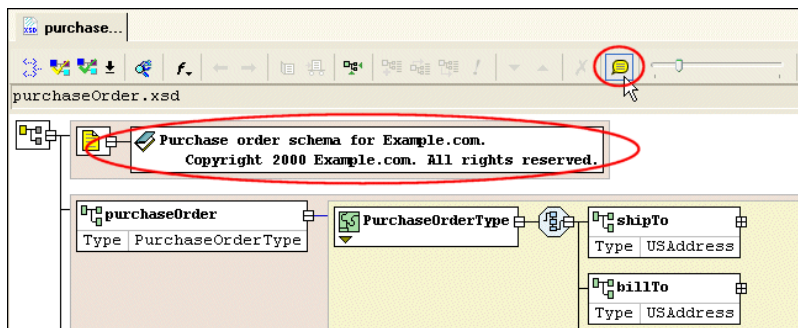




Figure 58. Show Documentation with the Click of a Button


Moving around the diagram


There are several ways to move around the diagram pane:

- To move from node to node in the diagram, press the arrow keys.

- You can use the scroll bars to explore the diagram; the zoom slider lets you change the magnification.
- Click **Go to Definition** () on the shortcut menu to display a new page that shows just the type definition.
- Click **Display Definition** () on the shortcut menu to jump to the place in the XML Schema where the type is defined.

Text Pane

The text pane appears directly beneath the XML Schema diagram pane. It displays the XML Schema code represented by the nodes you create in the diagram. The default font is Courier New, but you can change it to whatever font you want by clicking the **Change Font** button ().

Stylus Studio synchronizes the diagram and text views of the XML Schema – any changes you make in the diagram are reflected in the text pane, and vice versa. Synchronization information is displayed in the bar that separates the diagram and text panes. Current status is displayed on the right. When the two views are synchronized, Stylus Studio displays this graphic: . When Stylus Studio detects a change, such as a change to the text, it displays a message and changes the status graphic, as shown in [Figure 59](#).

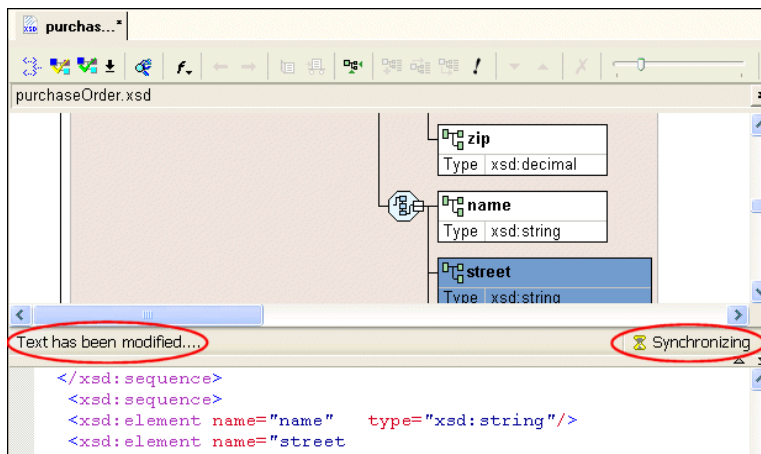


Figure 59. Synchronization Status is Monitored and Displayed

Stylus Studio also flags any XML Schema errors in the text pane – lines that contain errors are identified with a red dot, and the type and location of the error is displayed in the status area at the top of the text pane, as shown here:

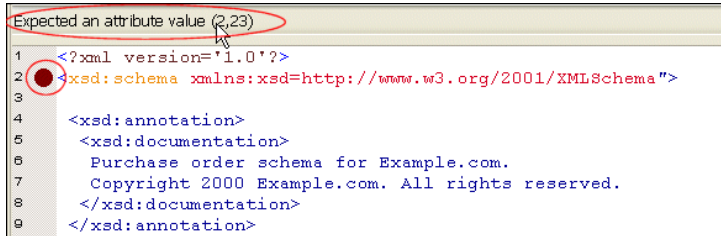


Figure 60. Text Pane Highlights XML Schema Errors

When you click the error message, Stylus Studio jumps to that part of the XML Schema containing the error. When you correct one error, information about the next error detected by Stylus Studio (if any) is displayed in the status area.

You can use the splitter to resize the text pane to view more or less text, or you can hide it entirely using the controls on the splitter's right side.

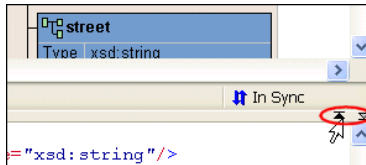


Figure 61. Splitter Controls Change Size of Text and Diagram Panes

Stylus Studio supports back-mapping between the text pane and the XML Schema diagram pane – if you click a node in the diagram, Stylus Studio scrolls the text pane to display the line of XML Schema that defines the node you clicked. A blue triangle is displayed to the left of the exact line of code.

Definition Browser

The definition browser is a drop-down list that displays all the child nodes of the schema node. It is located at the top of the **Diagram** tab.

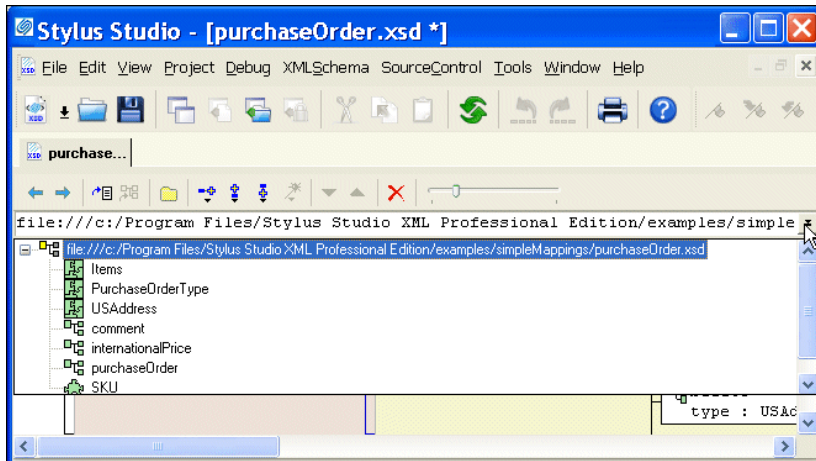


Figure 62. Definition Browser

When you select a node from the definition browser, Stylus Studio displays a new page in the XML Schema diagram pane that shows the definition of the node you select. In addition, the definition browser displays information about that node.

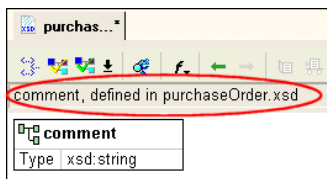



Figure 63. Information About the Selected Node

To return to the page you were viewing previously, press the back () button.

Note When you display a node using the definition browser, the focus of the text pane does not change. Clicking the node jumps you to that part of the XML Schema where the node is defined.

Editing Tools of the Diagram Tab


Many of the operations you perform in the **Diagram** tab can be performed in a number of ways. This section briefly describes menu and tool bar use, and introduces additional features for defining XML Schema.

This section covers the following topics:

- “[Menus and Tool Bars](#)” on page 78
- “[In-place Editing](#)” on page 79
- “[Drag-and-Drop](#)” on page 79
- “[QuickEdit](#)” on page 80
- “[Refactoring](#)” on page 81

Menus and Tool Bars

The complete set of available operations is defined by the menu system. The tool bar provides a subset of frequently performed operations. The top-level menu (**XMLSchema > Diagram**), the shortcut menus, and the tool bar are context sensitive – only operations that are permitted given the current context are available. For example, if you want to add an element to a sequence, you can

- Select **XML Schema > Diagram > Add > Element** from the main menu
- Select **Add > Element** from the sequence shortcut menu
- Click the **Add** button  on the tool bar and select **Element** from the drop-down list it displays

Each of these actions lets you add a *new* node, in this case, an element, to your XML Schema definition.

In-place Editing

In-place editing allows you to change node names and properties directly in the diagram. For example, say you want to change the value of the **Mixed** property of the `PurchaseOrderType` complexType. Just double-click the property. Stylus Studio opens the property for editing, as shown in [Figure 64](#).

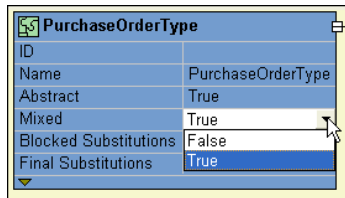


Figure 64. In-Place Editing

Similarly, if you double-click a node name, Stylus Studio places the property in edit mode, allowing you to type a new name.

Tip To display all of a node's properties in the diagram, see "[Displaying Properties](#)" on page 72.

Drag-and-Drop

An alternative to using the menu and the tool bar is to use *drag-and-drop*, which lets you add an *existing* node to another node's definition. For example, say you wanted to add an existing element to a sequence. You can do this by dragging the element icon to the sequence icon, as shown in [Figure 65](#).

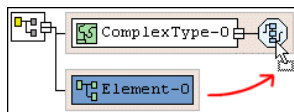


Figure 65. Using Drag-and-Drop to Define a Node

Use drag-and-drop any time you want to define a node using a node already defined in your XML Schema.

Tip When you drag and drop, you remove the element from its current context. If you want to make a *copy* an element, press and hold the CTRL key when you perform the drag operation.

Typical targets of drag-and-drop operations include the following nodes

- schema
- sequence
- choice
- all
- list
- annotation
- restriction
- union

Typical sources for drag-and-drop operations include the following nodes

- simpleType
- element
- annotation

Tip Any node you drag to the schema node is created as a child of the schema node.

QuickEdit

QuickEdit is a feature of the **Diagram** tab that streamlines common editing operations. For example, you can use QuickEdit to

- Change a sequence to a choice or to an all
- Specify a restriction for a simpleType
- Create sequence, choice, any, and all element definitions

For example, the following structure was created by selecting QuickEdit > Add Elements Choice from the complexType's shortcut menu.

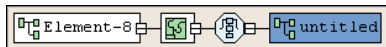



Figure 66. QuickEdit Creates Complex Definitions With a Click

QuickEdit appears on the top-level and shortcut menus in those contexts in which it is available, and it is also available on the tool bar by pressing the QuickEdit button .

Refactoring

Refactoring is a process that allows you to copy globally defined nodes from one XML Schema and paste them in a new XML Schema. The difference between refactoring and a simple copy is that refactoring includes both the node you select and all its dependencies. Consider the following example: here is how the purchaseOrder node appears when it is *copied* from purchaseOrder.xsd and pasted into a new XML Schema document:

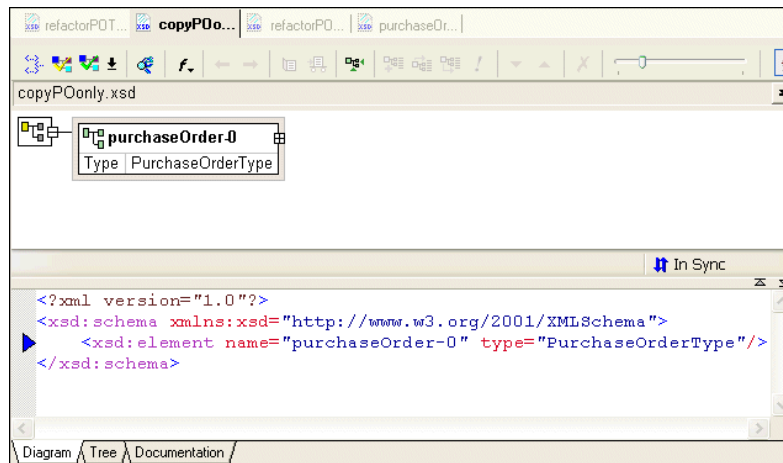


Figure 67. Simple Copy/Paste of a Node

As you can see, the copy action copies *only* the selected node to the clipboard. When the same node is copied using refactoring and pasted into another XML Schema document, the node, and all its dependencies copied as well.

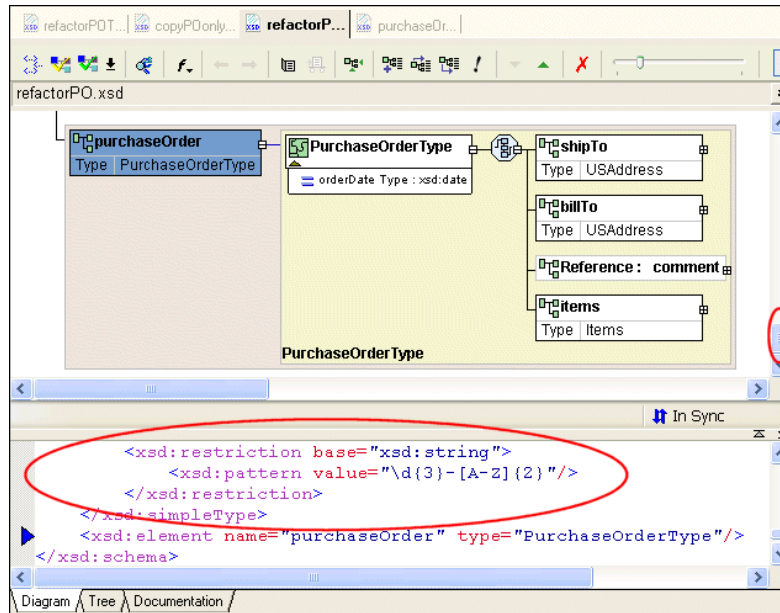


Figure 68. Refactoring Copy/Paste of a Node

Not all of the diagram or text are displayed in this illustration, but it is clear that more than just the purchaseOrder node was copied to the clipboard. For example, the purchaseOrder's type, PurchaseOrderType complexType has been copied, as well as PurchaseOrderType's element and sequence nodes, such as shipTo, billTo, and items.

If you were to scroll up either the text pane or the diagram pane, you would also see, for example, the complete definitions for other global complexTypes such as SKU and USAddress.

◆ **To refactor a node:**

1. Right-click the node you want to refactor.
2. Select **Refactoring > Copy** from the node's shortcut menu.

Note If the node is not globally defined, refactoring is not available.

The node and all its dependencies are copied to the clipboard.

3. To paste the node in the target XML Schema document, select **Refactoring > Paste** from the shortcut menu.

Description of Sample XML Schema

Suppose you want to define an XML Schema that defines book, magazine, and newsletter elements. The type of each of these elements is `PublicationType`. The XML Schema defines the `PublicationType` complexType. An element that is a `PublicationType` has the following description:

- The `genre` attribute specifies the style of the publication. That is, whether it is a book, magazine, or newsletter.
- There is always exactly one `title` element.
- The `subtitle` element is optional.
- There must be at least one author element and there can be more. Each author element contains one `first-name` element and one `last-name` element.
- Of the following three elements, exactly one must always be present:
 - `ISBNnumber`
 - `PUBnumber`
 - `LOCnumber`
- The elements must be in the order specified in this list.

The following topics in this section describe how to define this XML Schema using the **Diagram** tab of the XML Schema Editor.

Defining a complexType in a Sample XML Schema in the Diagram View

The steps for defining the `PublicationType` complexType described in “[Description of Sample XML Schema](#)” on page 83 are presented in the following topics:

- “[Defining the Name of a Sample complexType in the Diagram View](#)” on page 84
- “[Adding an Attribute to a Sample complexType in the Diagram View](#)” on page 85
- “[Adding Elements to a Sample complexType in the Diagram View](#)” on page 86
- “[Adding Optional Elements to a Sample complexType in the Diagram View](#)” on page 87
- “[Adding an Element That Contains Subelements to a complexType in the Diagram View](#)” on page 87

- “Choosing the Element to Include in a Sample complexType in the Diagram View” on page 89

Defining the Name of a Sample complexType in the Diagram View

◆ To define a complexType in a sample XML Schema:

1. From the Stylus Studio menu bar, select **File > New > XML Schema**.
Stylus Studio displays the XML Schema Editor. Maximize the XML Schema Editor window. If the **Project** window is visible, you can close it.
2. At the bottom of the XML Schema editor, click the **Diagram** tab.
Stylus Studio displays the **Diagram** view for the new schema.

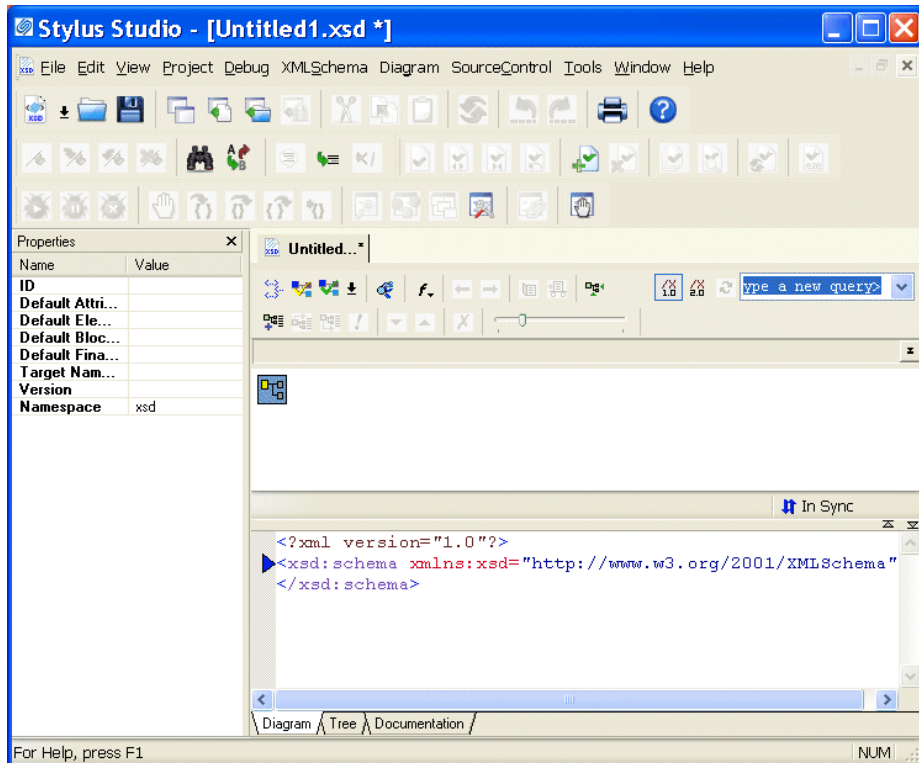


Figure 69. The XMLSchema Editor Diagram Tab

3. Right-click the schema node in the XML Schema diagram pane and select **Add > ComplexType** from the shortcut menu.

Alternatives: This action is also available from the **XMLSchema > Grid Editing** menu. Stylus Studio displays a representation for the new node in the diagram. The complexType properties appear in the **Properties** window. The new complexType has a default name of ComplexType-0.

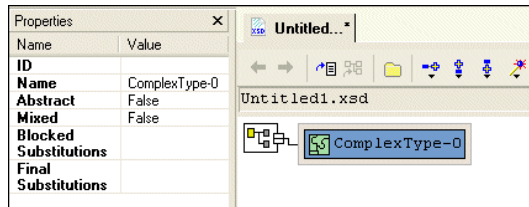



Figure 70. New complexType

4. Type `PublicationType` in the **Name** property in the **Properties** window and press Enter.
Stylus Studio updates the diagram and the XML Schema in the text pane.
5. Click **Save** .
6. In the **Save As** dialog box, in the **URL** field, type `bookstoreDiagram.xsd`, and click **Save**. You can save it in the `examples` directory of the Stylus Studio installation directory or in a directory of your choice.

Adding an Attribute to a Sample complexType in the Diagram View

◆ **To add the genre attribute to the PublicationType complexType:**

1. Right-click the `PublicationType` node.
2. In the shortcut menu that appears, select **Add > Attribute**.
Stylus Studio displays a node for the new attribute (`untitled`).

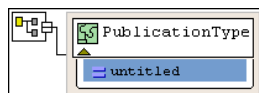


Figure 71. Adding an Attribute in the Diagram Tab

3. In the **Properties** window, type `genre` as the name of the new attribute and press Enter.
4. In the **Properties** window, click the **Type** field.
Stylus Studio displays a drop-down list of built-in types.

5. Scroll down to **xsd:string** and click it, or type `xsd:string` and press Enter. The diagram should now look like the one shown in [Figure 72](#).



Figure 72. The Finished genre Attribute


Tip You can toggle the display of attributes by clicking the small triangle at the bottom of the complexType node.

6. Click **Save** .

Adding Elements to a Sample complexType in the Diagram View

The elements belonging to this complexType must occur in a specific order. Before defining the first element, you need to create a sequence node to define this requirement in the XML Schema.

◆ **To add the `title` element to the `PublicationType` complexType:**


1. Right-click the `PublicationType` node.
2. In the shortcut menu that appears, select **Add > Sequence**.
The sequence node is added to `bookstoreDiagram.xsd`. The sequence modifier indicates that if an instance document contains the sequence node's child elements (the elements you will add next), they must be in the order in which they are defined.
3. Type `title` and press Enter.
4. Right-click the sequence node .
5. In the shortcut menu that appears, select **Add > Element**.
A child element is added to the `PublicationType` node.
6. In the **Properties** window, click the **Name** field and enter `title`.
7. In the **Properties** window, click the **Type** field.
Stylus Studio displays a drop-down list of built-in types.
8. Scroll down to **xsd:string** and click it, or type `xsd:string` and press Enter.

According to the XML Schema requirements described earlier, the `title` element can occur only once. By default, the default value for the `Min Occur`. (minimum occurrences)

and `Max Occur.` (maximum occurrences) properties is 1. You want exactly one instance of the `title` element in `PublicationType`, so you can accept these defaults.

Adding Optional Elements to a Sample complexType in the Diagram View

◆ **To add the optional `subtitle` element to the `PublicationType` complexType:**

1. Right-click the sequence node .
2. In the shortcut menu that appears, select **Add > Element**.
Below the `title` element, Stylus Studio displays a rectangle for the new element definition.
3. Rename the new element `subtitle`.
4. Give the new element a data type of `xsd:string`.
5. Give the new element a minimum occurrences value of 0.
You can accept the default of 1 for the **Max Occur.** property.

6. Click **Save** .

At this point, the XML Schema diagram should look like [Figure 73](#):

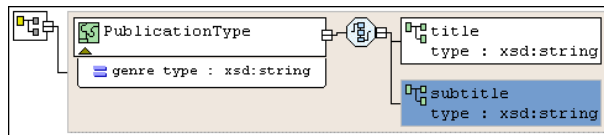



Figure 73. `PublicationType` complexType

Adding an Element That Contains Subelements to a complexType in the Diagram View

The sample schema requirements (see [“Description of Sample XML Schema”](#) on page 83) state that the `PublicationType` complexType must include at least one `author` element. Further, an `author` element must include a `first-name` element and a `last-name` element.

Each element that can contain one or more subelements is a complexType. Consequently, to add the `author` element to the `PublicationType` complexType, you must first define the `AuthorType` complexType. You can then add an element that is of `AuthorType` to the `PublicationType` complexType.

◆ **To define the AuthorType complexType:**

1. Right-click the schema node in the XML Schema diagram pane and select **Add > Complex Type** from the shortcut menu.
Alternatives: This action is also available from the **XMLSchema > Grid Editing** menu. Stylus Studio displays a representation for the new node in the diagram. The complexType properties appear in the **Properties** window.
2. Type AuthorType in the **Name** property in the **Properties** window and press Enter. Stylus Studio updates the diagram and the XML Schema in the text pane.
3. Right-click the AuthorType node in the diagram.
4. In the shortcut menu that appears, select **Add > Sequence**. Stylus Studio displays the sequence node .
5. Right-click the sequence node.
6. In the shortcut menu that appears, select **Add > Element**.
7. Type first-name in the **Name** property in the **Properties** window and press Enter.
8. Change the **Type** property to xsd:string and press Enter.
9. Repeat [Step 5](#) through [Step 8](#) to add a new element to the sequence, using last-name as the name of the new element.

◆ **Now you can add the author element to the PublicationType complexType:**

1. Right-click the sequence node belonging to the PublicationType node.
2. In the shortcut menu that appears, select **Add > Element**. Stylus Studio displays a representation for the new node in the diagram. The complexType properties appear in the **Properties** window.
3. Type author in the **Name** property in the **Properties** window and press Enter. Stylus Studio updates the diagram and the XML Schema in the text pane.
4. Click the **Type** field in the **Properties** window. Stylus Studio displays a drop-down list of built-in types plus any types you have defined, such as the AuthorType you defined in the previous procedure.

5. Select **AuthorType** from the drop-down list.

Tip A plus sign appears on the right side of the author element. You can click the plus sign to display the definition of the AuthorType complexType, which was just added to the author element.

6. Click the **Max Occur.** field.
7. In the drop-down list that appears, click **unbounded**.

Tip When you give an element an unbounded maximum number of occurrences, Stylus Studio renders the node using two outlines, to indicate that multiple occurrences of this element are allowed.

8. Click **Save** .

At this point, the XML Schema diagram should look like [Figure 74](#):

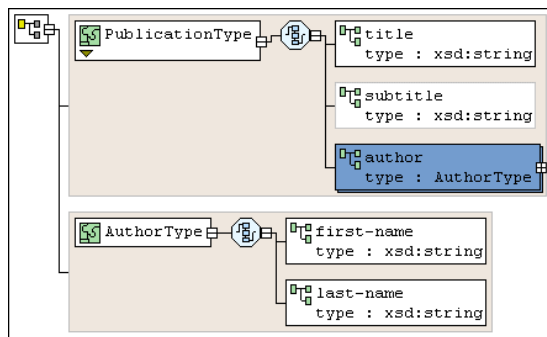


Figure 74. author Element with AuthorType complexType

Choosing the Element to Include in a Sample complexType in the Diagram View

In the sample XML Schema, you want `PublicationType` elements to contain an `ISBNnumber`, `PUBnumber`, or `LOCnumber` element.

◆ **To specify this:**


1. Right-click the sequence node belonging to the `PublicationType` node.
2. In the shortcut menu that appears, select **Add > Sequence**.

Stylus Studio displays a representation for the new sequence node in the diagram. Sequence properties appear in the **Properties** window.

We added the sequence node in error – the specification requires that this node be a choice node. The QuickEdit feature makes it easy to correct errors such as this.

3. Right-click the new sequence node. In the shortcut menu that appears, select **QuickEdit > Switch to Choice**.

Stylus Studio changes the sequence node to the choice node ().

4. Right-click the new choice node.
5. In the shortcut menu that appears, select **Add > Element**.
6. In the **Properties** window, change the **Name** to ISBNnumber and press Enter.
7. In the **Properties** window, change the **Type** xsd:int and press Enter.
8. Repeat [Step 4](#) through [Step 7](#) twice: once to add the PUBnumber element, and once to add the LOCnumber element.
9. Click **Save** .

The definition of the `PublicationType` complexType is now complete and should look like [Figure 75](#):

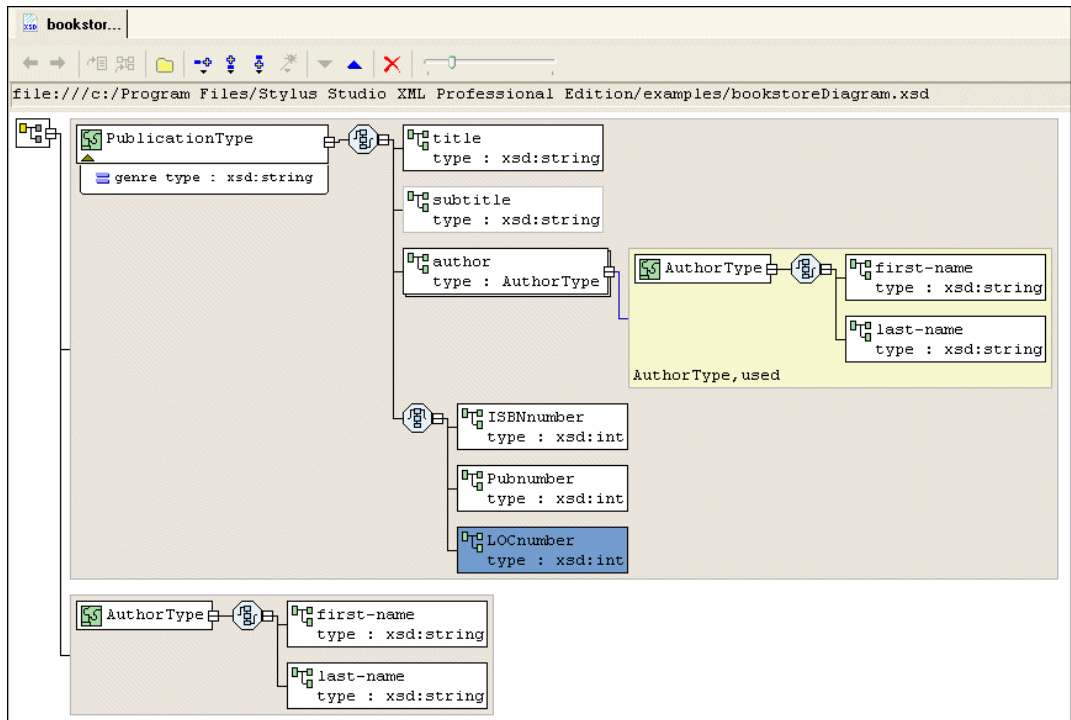



Figure 75. PublicationType complexType Fully Defined

Defining Elements of the Sample complexType in the Diagram View

In the final step of defining `bookstoreDiagram.xsd`, you define elements that are of the `PublicationType` complexTypes you defined earlier – `book`, `magazine`, and `newsletter` elements.

◆ **To define the `book`, `magazine`, and `newsletter` elements in the sample XML Schema:**

1. Right-click the schema node in the diagram.
2. In the shortcut menu that appears, select **Add > Element**.
Stylus Studio displays a node for the new element in the XML Schema diagram pane. The properties for the new element appear in the **Properties** window.

3. Type `book` as the name of the new element and press Enter.
 4. In the **Properties** window, click the **Data Type** field. Stylus Studio displays a drop-down list of built-in types plus any types you have defined.
 5. Click **PublicationType**.
 6. Repeat [Step 1](#) through [Step 5](#) twice: once to add the `magazine` element, and once to add the `newsletter` element.
 7. Click **Save** .
- The `bookstoreDiagram.xsd` document is now complete.
8. Select **XMLSchema > Validate Document** from the menu to validate the XML Schema document you created.
- The validation message appears in the **Output** window, as shown in

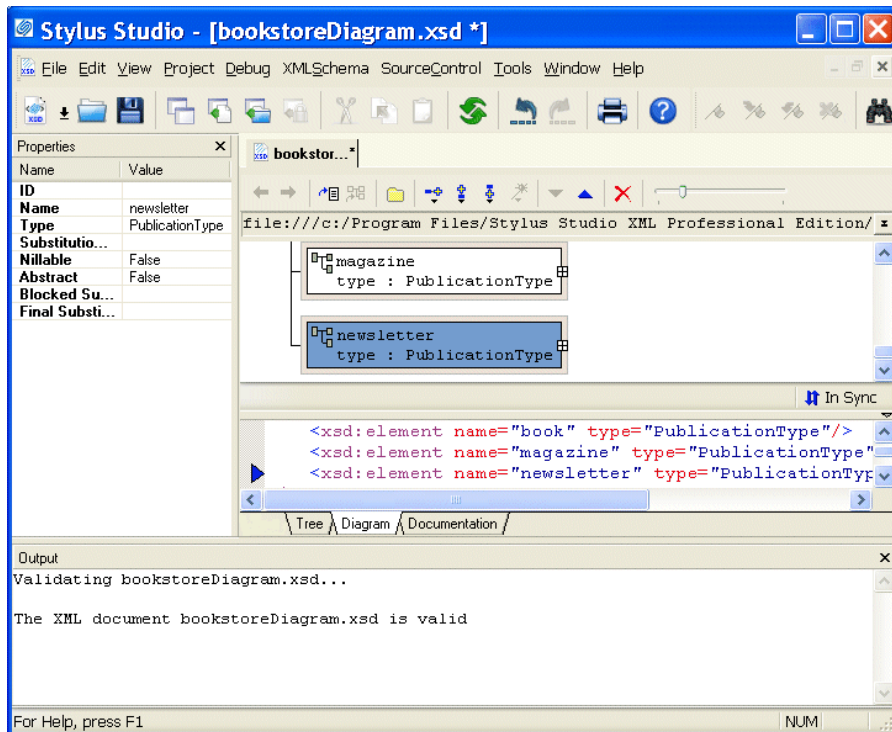


Figure 76. Validation of bookstoreDiagram.xsd

Opening Files in Stylus Studio

This section describes the types of files Stylus Studio recognizes, how to add new file types to Stylus Studio, and how to open files using the Stylus Studio File Explorer and other methods. This section covers the following topics:

- “Types of Files Recognized by Stylus Studio” on page 93
- “Using the File Explorer” on page 95
- “Dragging and Dropping Files in the Stylus Studio” on page 98
- “Other Ways to Open Files in Stylus Studio” on page 99
- “Adding File Types to Stylus Studio” on page 100

Tip You can set an option so that when you open Stylus Studio, Stylus Studio automatically opens any files that were open the last time you closed Stylus Studio. See “Options - Application Settings” on page 1129.

Types of Files Recognized by Stylus Studio

Stylus Studio recognizes over ten types of files by default. Each file is associated with a Stylus Studio module, or editor, appropriate for its type, as shown in the following table.

Table 2. File Extensions and Associated Modules

<i>File Name Extension</i>	<i>Stylus Studio Module</i>
.conv	Custom XML Conversion Editor
.dff	XML Diff Viewer
.dtd	DTD Schema Editor
.java	Java Debugger
.pipeline	XML Pipeline
.prj	Project framework
.report	XML Publisher
.wscc, .wsc	Web Service Call Composer
.xml	XML Editor
.xquery	XQuery Editor

Table 2. File Extensions and Associated Modules

<i>File Name Extension</i>	<i>Stylus Studio Module</i>
.xsd	XML Schema Editor
.xsl, .xslt	XSLT Stylesheet Editor

You can add your own file types to this list, specify the module you want them opened in, and, optionally, specify Stylus Studio as the default application for viewing and editing files of that type. See [“Adding File Types to Stylus Studio”](#) on page 100.

Opening Unknown File Types

When you try to open a file of a type that is not recognized by Stylus Studio, Stylus Studio displays the **Choose Module for** dialog box, which allows you to specify the module or editor you want to use to open the file.

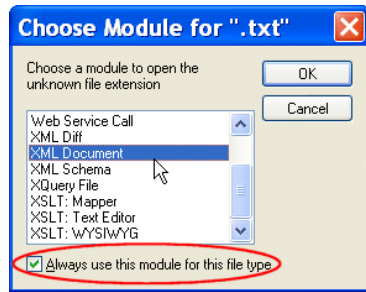


Figure 77. Choose Module Dialog Box

When you respond to this dialog box, you can optionally indicate whether you want all files of that type to be associated with the Stylus Studio module you select in the future. File types you associate with a Stylus Studio module are added to the **File Types** page of the **Options** dialog box. You can remove or change the module association at any time. See [“Adding File Types to Stylus Studio”](#) on page 100 for more information.

Opening Files Stored on Third-Party File Systems

Stylus Studio provides access XML documents stored on third-party file systems like Raining Data® TigerLogic® XML Data Management Server (TigerLogic XDMS).

See [Integrating with Third-Party File Systems](#) on page 1007 for more information.

Modifications to Open Files

If a file that is open in Stylus Studio is modified (changed, and saved) outside Stylus Studio, Stylus Studio alerts you that the file has been changed and gives you the chance to reload it.

Using the File Explorer

The Stylus Studio File Explorer is a dockable window that provides easy access to any file system accessible from the computer on which you are running Stylus Studio. You can use the File Explorer to quickly add files to Stylus Studio and open files in Stylus Studio, as well as to perform typical file management tasks (like renaming and deleting files, for example).

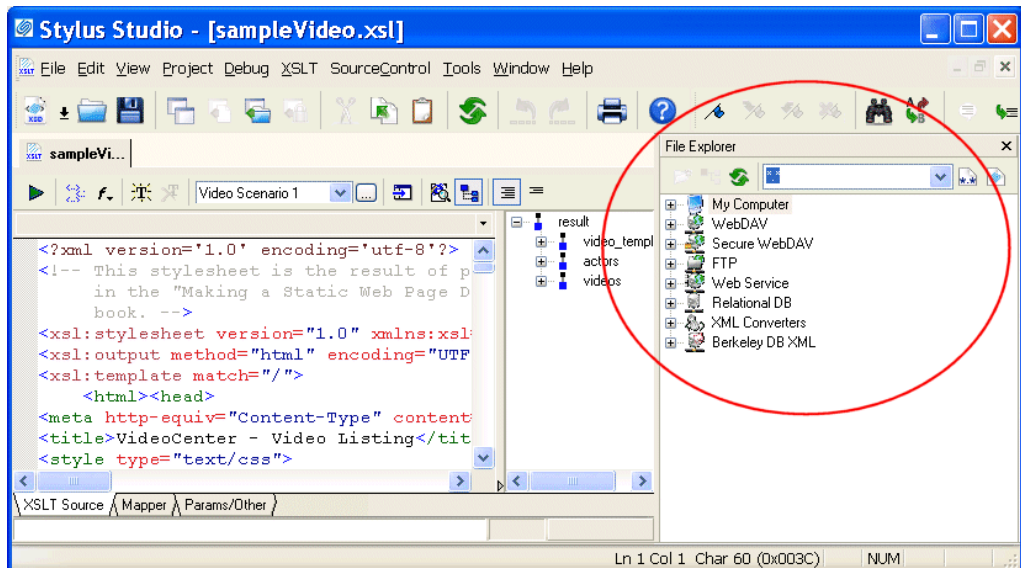


Figure 78. Stylus Studio File Explorer

By default, the **File Explorer** window appears on the right side of the Stylus Studio window, but you can drag it anywhere on your desktop. You can close/open the **File Explorer** window from the **View** menu.

How to Use the File Explorer to Open Files

There are several ways to open files using the File Explorer:

- Double-click the file
- Right-click the file and select **Open** or **Open With** from the shortcut menu

Tip

Open With allows you to select the module you want to use to open the file.

- Drag and drop the file. See [Dragging and Dropping Files in the Stylus Studio](#) on page 98.

When you open a file by double-clicking or using the **Open** shortcut menu, Stylus Studio opens the file in the module associated with the file type (the XML Editor for .xml files, for example). If the file type is not currently registered with Stylus Studio, you can register the file at this time using the **Choose Module for** dialog box. See “[Types of Files Recognized by Stylus Studio](#)” on page 93 for more information about file type/module associations in Stylus Studio.

Other Features of the File Explorer

The tool bar in the **File Explorer** window has several features that can help you navigate the file systems associated with your computer and work with individual documents.

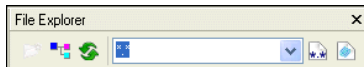







Figure 79. File Explorer Tool Bar

These features are summarized in the following table.

Table 3. File Explorer Tools

Tool	Description
 New Folder	Creates a new folder as a child of the folder with current focus. The default folder name is New Folder.
 Read Document Structure	Displays the structure of XML documents in tree form. You can drag exposed nodes onto the document tab area to open the document associated with that node. If you drag a node into an existing XQuery or XSLT document, Stylus Studio creates the document function with the XPath expression for that node. For example, if you drag the <code>title</code> element from <code>books.xml</code> into an XQuery document, Stylus Studio builds the following function: <code>doc("file:///c:/Program Files/Stylus Studio 2007 XML Enterprise Suite/examples/simpleMappings/books.xml")/books/book/title</code>
 Refresh	Refreshes the File Explorer window.
 Reset Filters	Resets the File Explorer filter from its current content to the wildcard (*.*) .
 Stylus File Types	Changes the File Explorer filter to display only file types associated with Stylus Studio: .xml, .xsd, .dtd, .java, .conv, and others.

Working with the File Explorer Filter

By default, the **File Explorer** window uses a wildcard filter to display all file types (*.*) .

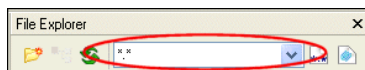



Figure 80. File Explorer Filter

You can

- Type your own filter (*.txt, for example). If you want to use multiple filters, separate them with a semicolon (*.txt; *.html, for example).
- Use the **Stylus File Types**  button to change the filter to display only file types associated with Stylus Studio (.xml, .xsd, .dtd, .java, .conv, and others)

Stylus Studio remembers the filters you create and adds them to the drop-down list.

Dragging and Dropping Files in the Stylus Studio

You can open a file in Stylus Studio by dragging the file from the File Explorer (or other file system browsers, like Windows Explorer) and dropping it inside Stylus Studio. How Stylus Studio behaves depends on where you drop the file, as summarized in [Table 4](#).

Table 4. How Stylus Studio Handles Dragged-and-Dropped Files

<i>If You Drop the File Here</i>	<i>Stylus Studio Does This</i>
On the document editor area (when no documents are open)	Opens the document editor associated with the type of file you selected. See Types of Files Recognized by Stylus Studio on page 93. If the file type is not currently registered with Stylus Studio, you can register the file at this time using the Choose Module for dialog box. See Opening Unknown File Types on page 94.
On the document editor tab area	Opens the document editor associated with the type of file you selected. See Types of Files Recognized by Stylus Studio on page 93. If the file type is not currently registered with Stylus Studio, you can register the file at this time using the Choose Module for dialog box. See Opening Unknown File Types on page 94.
In an active document	Adds the file's URL to the end of the document.
On another file in the File Explorer	Performs the operation associated with the target file and opens the resulting document in its own editor. For example, you might use this operation to convert a text file to XML by dropping the .txt file on a converter file (.conv).
In a project in the Project window	Adds the file to the project. If the file type is not currently registered with Stylus Studio, you can register the file at this time using the Choose Module for dialog box. See Opening Unknown File Types on page 94.

Other Ways to Open Files in Stylus Studio

In addition to the File Explorer and drag-and-drop, you can also open files in Stylus Studio from the following places:

- The **Open** dialog box (displayed when you select **File > Open** from the menu, for example). By default, Stylus Studio opens the file in the editor associated with files of the type you select (see [Types of Files Recognized by Stylus Studio](#) on page 93). If you want, you can choose a different editor – you might want to open an XSLT stylesheet in the XML Editor, for example – when opening files from the **Open** dialog box.

To specify a different module, click the down arrow to the right of the **Open** button and select the module you want to use from the drop-down list as shown in [Figure 81](#).

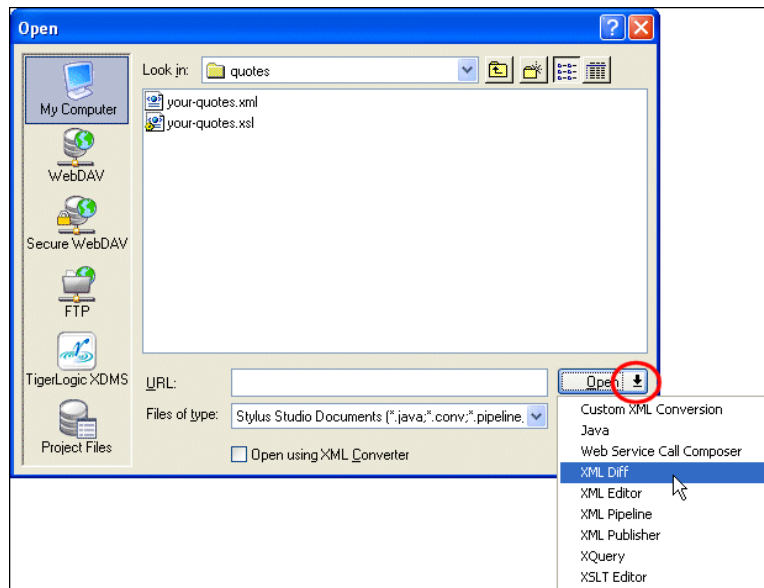


Figure 81. Choose the Module to Use When Opening a File

- **Project** window – either double-click the file, or select **Open** or **Open With** from the file’s shortcut menu.
- Other file system browsers (like Windows Explorer, for example) – for files recognized by Stylus Studio, just double-click the file. See [“Opening Unknown File Types”](#) on page 94.

Adding File Types to Stylus Studio

You use the procedure described in this section to associate a file type (.txt, for example) with a specific Stylus Studio module or editor. Once you do this, any time you open a file of that type from within Stylus Studio (using the File Explorer, for example), that file is opened in the editor you specify.

You can optionally specify that you want to use Stylus Studio as the default editor for files of this type, regardless of where the file is opened (from a file browser like Total Commander, for example).

Note You do not need to specify the usual extensions, such as xml, xs1, and java. Use the procedure described in this section for file name extensions peculiar to your application or environment.

◆ **To add a file type to Stylus Studio:**

1. From the Stylus Studio menu bar, select **Tools > Options**.
Stylus Studio displays the **Options** dialog box.
2. Under **General**, click **File Types**.
The **File Types** page appears.

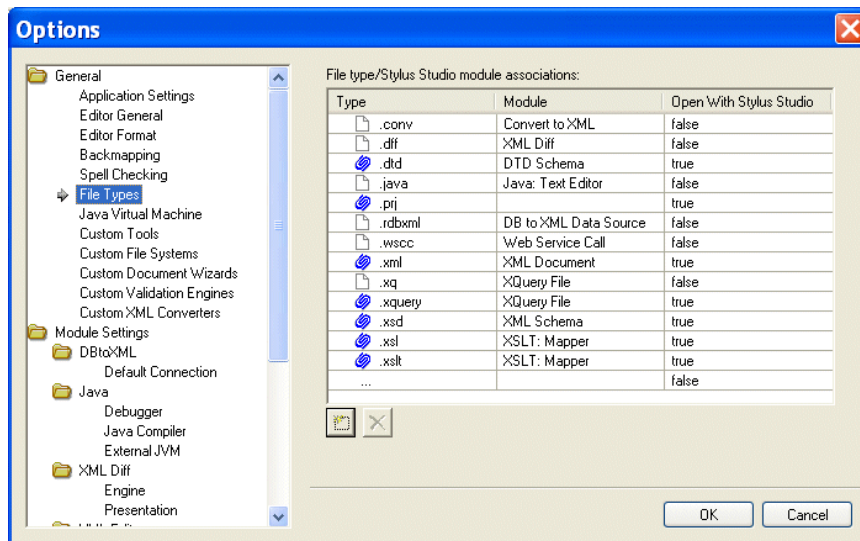



Figure 82. Associating File Types with Stylus Studio Editors

3. To add a new file type/module association, click the **Add**  button.
Alternative: Double-click the **Type** field.
4. Type the new extension, including the period, and press Enter.
Stylus Studio adds the file type and selects a default module in the **Module** field.
5. Optionally, select a different module using the drop-down list in the **Module** field.
6. If you want to always use Stylus Studio to open files of this type, change the value in the **Open with Stylus Studio** field to **True**.
7. To add another file type, repeat [Step 3](#) through [Step 6](#).
8. When you are done, click **OK**.

Deleting File Types

◆ To delete a file type:

1. Click the file type you want to delete.
2. Click the **Delete**  button.
3. Click **OK**.

Working with Projects

A *project* in Stylus Studio is a group of files related to a given XML application. A project might include XML, XML Schema, and XQuery files, as well as OASIS catalogs, for example. A project can contain subprojects, and subprojects can contain subprojects. The Stylus Studio project framework allows you to name projects (project files are saved with a .prj extension), and it provides several tools for managing the projects you create.

Projects are simply a convenience for organizing files – a file does not have to belong to a project in order for you to edit it in Stylus Studio. For example, Stylus Studio includes all sample application files in the `examples` project. You can find the `examples.prj` file in the `examples` directory of your Stylus Studio installation directory.

This section discusses the following topics:

- [“Displaying the Project Window”](#) on page 102
- [“Creating Projects and Subprojects”](#) on page 104
- [“Saving Projects”](#) on page 104

- “Opening Projects” on page 104
- “Adding Files to Projects” on page 105
- “Copying Projects” on page 106
- “Rearranging the Files in a Project” on page 107
- “Removing Files from Projects” on page 107
- “Closing and Deleting Projects” on page 107
- “Setting a Project Classpath” on page 108
- “Using Stylus Studio with Source Control Applications” on page 110

Displaying the Project Window

When you open Stylus Studio for the first time, Stylus Studio displays the **Project** window with the examples project. (The **File Explorer** window is displayed on the right.)

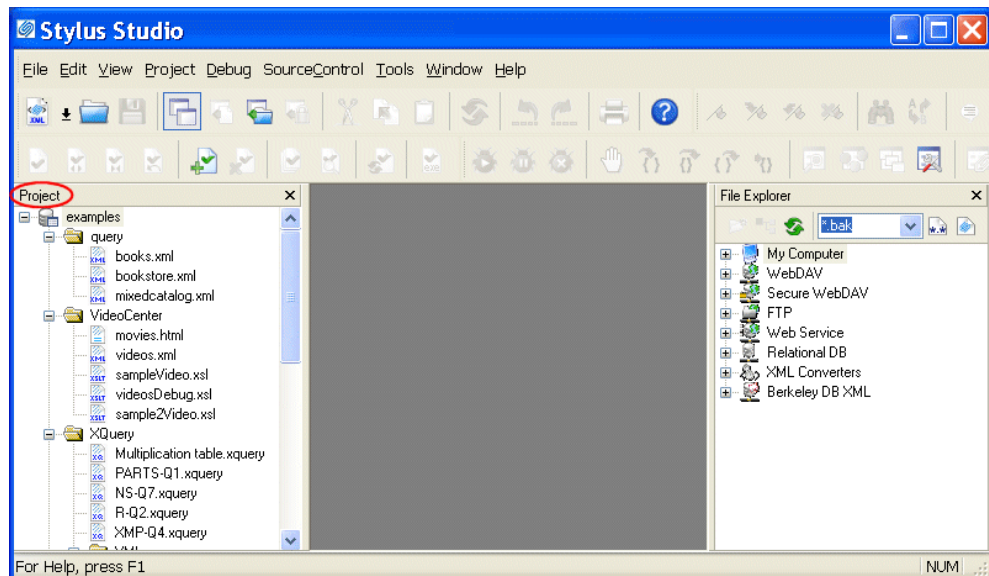



Figure 83. Project Window with Default Project Displayed

There are several ways to toggle the display of the **Project** window. You might want to close the **Project** window in order to gain more space in the editor you are working with, for example.

◆ **To toggle Project window display:**

- From the Stylus Studio menu, select **View > Project Window**.
- In the Stylus Studio tool bar, click **Toggle Project Window** .

Tip The **Project** window is dockable – you can move it anywhere on your desktop.

◆ **To hide Project window:**

Click the **X** in the upper right corner of the **Project** window.

Tip When you hide the **Project** window, any open files remain open.

Displaying Path Names

You can control whether the **Project** window displays absolute or relative path names for files in projects. The default display is relative names.

◆ **To toggle the way path names appear:**

1. Display the **Project** window.
2. In the **Project** window, right-click to display the pop-up menu.
3. Click **Show Full URL Info**.

Other Documents

Stylus Studio displays documents that are not associated with a project in the **Other Documents** folder, which appears after the last folder or document in the currently displayed project. In addition, when you remove a file from a project, it is placed in the **Other Documents** folder.

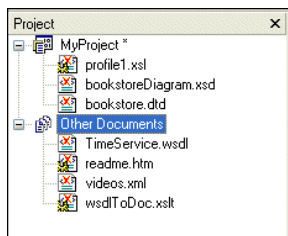


Figure 84. Other Documents Folder

You can add these documents to a project at any time. See [“Adding Files to Projects”](#) on page 105.

Creating Projects and Subprojects

You can create projects and organize any project into multiple levels of subprojects. You can add files to projects and save the project under a name you specify.

◆ **To create a project, select Project > New Project from the menu:**

Stylus Studio displays the new project in the **Project** window. The **Project** window displays information for only one project at a time.

◆ **To create a subproject:**

1. Right-click the project name, and click **New Project Folder** in the pop-up menu. Stylus Studio displays a default subproject folder name (NewFolder1, for example).
2. Type a new subproject name.
3. Press Enter.

There are several ways to add files to your projects and subprojects. See [“Adding Files to Projects”](#) on page 105.

Saving Projects

◆ **To save a project, select Project > Save Project.**

The first time you save a project, Stylus Studio prompts you to specify a name for your project. Stylus Studio appends .prj to the name you specify. It does not matter whether or not you specify the .prj extension. Stylus Studio does not allow a project to have any other file name extension.

When you save a project, references to the files part of the project are saved relative to the path of the project file. This allows you to move or share projects easily.

Opening Projects

You can have only one project open at one time. If you have a project open and you open a second project, Stylus Studio closes the first project and then opens the second project.

If the **Project** window is not visible when you open a project, Stylus Studio automatically displays the **Project** window.

◆ **To open a project:**

1. From the Stylus Studio menu bar, select **Project > Open Project**.
2. Navigate to and select your project file. For example, you can open `examples.prj` in the `examples` directory of your Stylus Studio installation directory. The `examples` project contains the files for all Stylus Studio sample applications.
3. Click the **Open** button.

Recently Opened Projects

Projects that were recently opened are displayed at the bottom of the **Project** drop-down menu. Click the project you want to open.

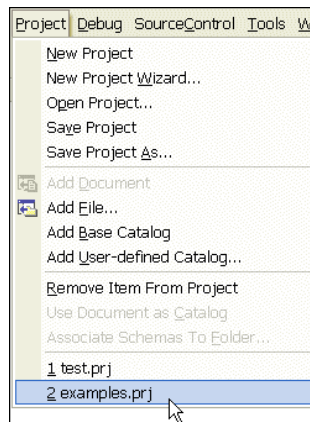


Figure 85. Recently Opened Projects Are Listed on the Project Menu

Adding Files to Projects

The easiest way to add a file to a Stylus Studio project is to drag the file from the File Explorer or another file browser (like Total Commander, for example) into the desired project folder. You can drag-and-drop multiple files at a time.


If the file type is unknown to Stylus Studio, the **Choose Module for** dialog box appears, which allows you to associate the file with a Stylus Studio module or editor. See [Opening Unknown File Types](#) on page 94 for more information.

Other Ways to Add Files to Projects

The following procedures describe other ways to add files to a project. Note that these procedures vary based on whether or not the file is already open in Stylus Studio.


When Files are Open in Stylus Studio

◆ **To add an open file to a project:**

1. Open the project to which you want to add the file.
2. Click the window (the Web Service Call Composer, for example) that contains the file you want to add.
3. In the Stylus Studio tool bar, click **Add Document to Project** .
Alternative: Select **Project > Add Document** from the Stylus Studio menu bar.

When Files are Closed

◆ **To add a closed file to a project:**

1. Open the project to which you want to add the file.
2. In the Stylus Studio tool bar, click **Add File to Project** .
Alternative: Select **Project > Add File** from the Stylus Studio menu bar.
The **Open** dialog box appears.
3. Navigate to the file you want to add and click the **Open** button.

Copying Projects


◆ **To copy a project:**

1. Open the project you want to copy.
2. From the Stylus Studio menu bar, select **Project > Save Project As**.
The **Save As** dialog box appears.
3. Navigate to the location for the project copy.
4. In the **URL:** field, type the name of the new project.
5. Click the **Save** button.

Rearranging the Files in a Project

The order in which files are displayed in the **Project** window has no effect on the project. You might want to place related files near each other, or place more frequently used project files toward the top of the project tree.


◆ **To rearrange files in a project:**

1. If the **Project** window is not visible, click **Toggle Project Window**  in the Stylus Studio tool bar.
2. In the **Project** window, click the file you want to move.
3. Drag it to its new location.

Removing Files from Projects

When you remove a file from a project, it is added to the **Other Documents** folder in the **Project** window.

◆ **To remove a file from a project:**

1. If the **Project** window is not visible, click **Toggle Project Window**  in the Stylus Studio tool bar.
2. In the **Project** window, click the path for the file you want to remove.
3. From the Stylus Studio menu bar, select **Project > Remove File from Project**.

Alternative: Press the Delete key.

Closing and Deleting Projects

Closing

◆ **To close a project, open another project or create a new project.**

Tip Toggling or closing the **Project** window does not close the project.

Deleting

- ◆ **To delete a project, remove its .prj file from the file system.**

Setting a Project Classpath

You can set a classpath at the project level. When Stylus Studio compiles or runs Java code, it always checks the project for a locally defined classpath first. If a project classpath has not been defined, Stylus Studio uses the classpath defined **Java Virtual Machine** page of the Options. dialog box.

Specifying Multiple Classpaths

You use the **Project Classpath** dialog box to specify one or more classpaths for a project. If multiple classpaths have been defined, Stylus Studio searches them in the order in which they are listed in the **Project Classpath** dialog box. You can use the up and down arrows at the top of this dialog box to change the classpath order.

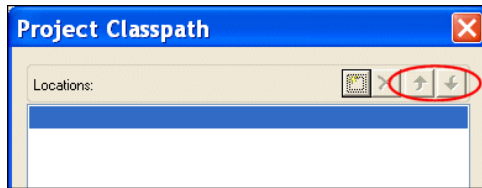


Figure 86. Use Arrow Buttons to Change the Order of Classpaths

How to Set a Project Classpath

- ◆ **To set a classpath for a project:**
 1. Open the **Project Window** if it is not already displayed.
 2. Right-click the project node.
The project shortcut menu appears.
Alternative: Select **Project > Set Classpath** from the Stylus Studio menu.
 3. Select **Set Project Classpath**.

The **Project Classpath** dialog box appears.

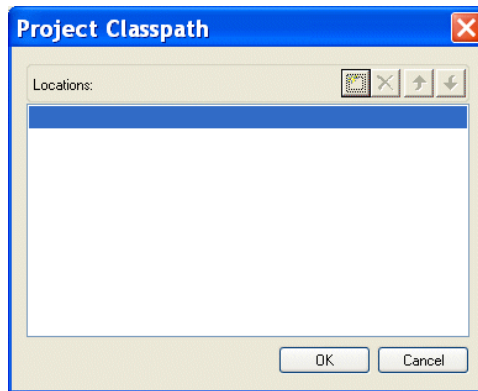




Figure 87. Project Classpath Dialog Box

4. Click the browse folders () button.
A new entry field appears in the **Locations** list box. Two buttons appear to the right of the entry field.
5. To add a JAR file to the classpath, click the browse jar files button ().
Stylus Studio displays the **Browse for Jar Files** dialog box.

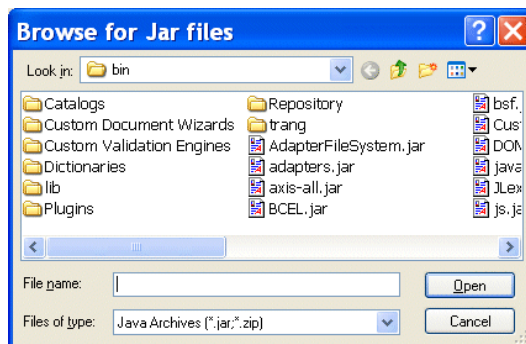


Figure 88. Browse for Jar Files Dialog Box

To add a folder to the classpath, click the browse folders button ()

Stylus Studio displays the **Browse for Folder** dialog box.

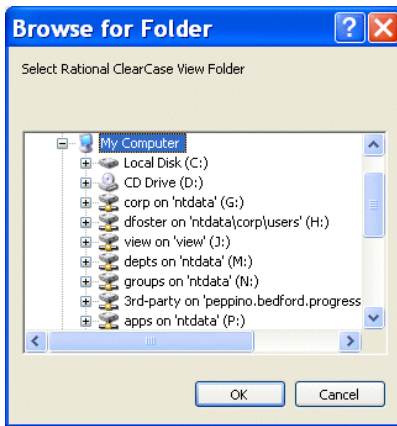


Figure 89. Browse for Folder Dialog Box

6. When you have located the JAR file or folder you want to add to the classpath, click OK.
7. Optionally, add other JAR files or folders to the classpath by repeating [Step 5](#) and [Step 6](#).

Using Stylus Studio with Source Control Applications

Stylus Studio supports the Microsoft Source Code Control Interface, allowing you to use Stylus Studio with any source code control system that supports the same interface used by Microsoft Visual Studio or Microsoft Visual Studio .NET.

Stylus Studio's source control support allows you to

- Add a file to source control
- Remove a file from source control
- Get the latest version of a source-controlled file
- Check out a file
- Check in a file
- Uncheck out a file
- Show the source control history of a file
- Show differences between versions of a file

In this section

This section covers the following topics:

- [“Tested Source Control Applications”](#) on page 111
- [“Prerequisites”](#) on page 111
- [“Using Stylus Studio with Microsoft Visual SourceSafe”](#) on page 112
- [“Using Stylus Studio with ClearCase”](#) on page 114
- [“Using Stylus Studio with Zeus CVS”](#) on page 117
- [“Specifying Advanced Source Control Properties”](#) on page 118

Tested Source Control Applications

Integration with the following source control applications has been tested:

- Microsoft Visual SourceSafe
- Clearcase/Attache
- CVS

Prerequisites

To use Stylus Studio’s source control features, you must have already installed the client software for your source control application, as shown in [Table 5](#).

Table 5. Working with Source Control Clients

<i>When Data Is In</i>	<i>You Need to Install</i>
SourceSafe repository	SourceSafe client or SourceOffSite
ClearCase	Attache client
CVS	Zeus-CVS product

In addition, files must belong to a Stylus Studio project before you can use them with a source control application.

Recursive Selection

When you build a project using files from a source control application, Stylus Studio gives you the option of recursively importing all projects that are subordinate to the project folder you select. This option, **Recursively import all subprojects**, appears on the **Build Project from SCC** dialog box, which appears when you start the New Project Wizard.

Selecting the **Recursively import all subprojects** option has the effect of selecting all the siblings of the selected file or directory, as well as any descendants of the selected item and its siblings. Stylus Studio creates a project that contains all files that Stylus Studio can open (for example, .xml, xsdt, and .xsd files) and that are in the directory hierarchy of the file or directory you select.

For example, suppose you check **Recursively import all subprojects**, and you select `c:\work\myproject\documentation.xml`. Stylus Studio creates a project that contains all Stylus Studio-editable files in `c:\work\myproject` and its subdirectories.

If you do not check **Recursively import all subprojects**, only the file you select is added to the new Stylus Studio project you create. You cannot select a directory if you do not select this option.

Using Stylus Studio with Microsoft Visual SourceSafe

◆ **To use Stylus Studio to operate on files that are under SourceSafe source control:**

1. From the Stylus Studio menu bar, select **Project > New Project Wizard**.

The **Project Wizards** dialog box appears.

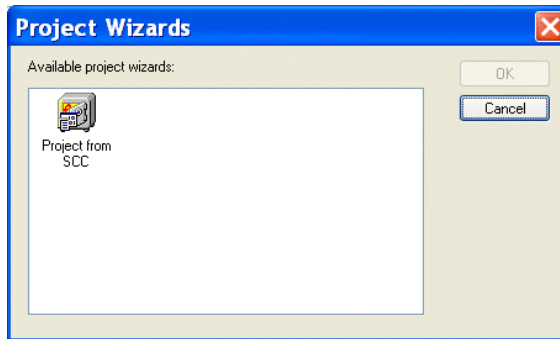


Figure 90. Project Wizards Dialog Box

2. Click **Project from SCC**, and click the **OK** button.
The **Build Project From SCC** dialog box appears.

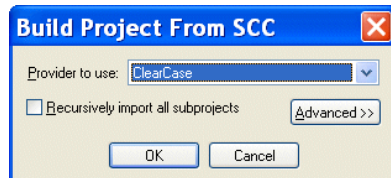


Figure 91. Build Project From SCC Dialog Box

3. Select **Microsoft Visual Sourcesafe** from the **Provider to use** drop-down list.
4. If you want to use Stylus Studio to access more than one file in a directory hierarchy, click the check box for **Recursively import all subprojects**. See [“Recursive Selection”](#) on page 111 if you need help with this step.
Depending on your installation, you might need to specify other properties. See [“Specifying Advanced Source Control Properties”](#) on page 118.
5. Click the **OK** button.
The **Visual SourceSafe Login** dialog box appears:

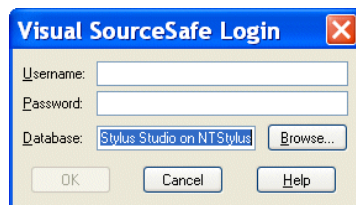


Figure 92. Visual SourceSafe Login Dialog Box

6. Specify the username and password; optionally, use the **Browse...** button to access a database other than the default database displayed in the **Database** field.
7. Click **OK**.

The **Create Local Project from SourceSafe** dialog box appears.

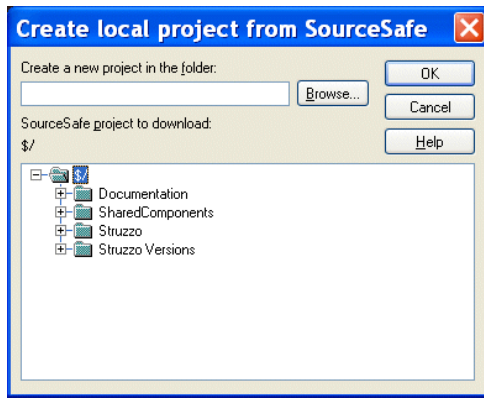


Figure 93. Create Local Project from SourceSafe Dialog Box

8. Select the folder in which you want to create the new project.
9. Click **OK**.

The project is created in Stylus Studio. A message displays the names of any files that were not added to the project because their extensions are not associated with a Stylus Studio editor.

Using Stylus Studio with ClearCase

◆ **To use Stylus Studio to operate on files that are under ClearCase source control:**

1. Use Attache to copy the files you want to work on from a ClearCase view to the local file system.

Note If you move these files from this directory after you create the project, you must specify the new directory that contains the files in the **Local Project Path** field of the **Source Control Properties** dialog box. To access this dialog box, select **SourceControl > Source Control Properties** from the Stylus Studio menu bar.

2. From the Stylus Studio menu bar, select **Project > New Project Wizard**.

The **Project Wizards** dialog box appears.

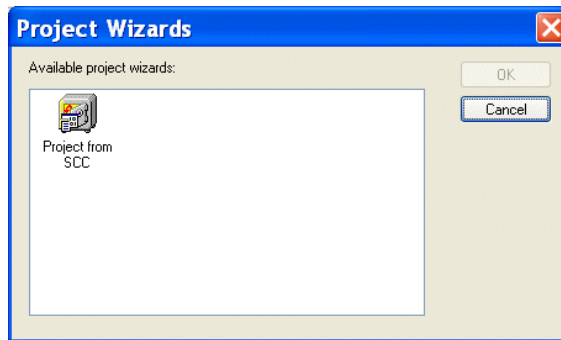


Figure 94. Project Wizards Dialog Box

3. Click **Project from SCC**, and click the **OK** button. The **Build Project From SCC** dialog box appears.

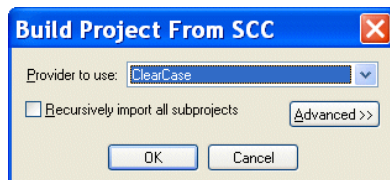


Figure 95. Build Project From SCC Dialog Box

4. Select **Clearcase** from the **Provider to use** drop-down list.
5. If you want to use Stylus Studio to access more than one file in a directory hierarchy, click the check box for **Recursively import all subprojects**. See [“Recursive Selection”](#) on page 111 if you need help with this step.
Depending on your installation, you might need to specify other properties. See [“Specifying Advanced Source Control Properties”](#) on page 118.
6. Click the **OK** button.

The **Browse for Folder** dialog box appears.

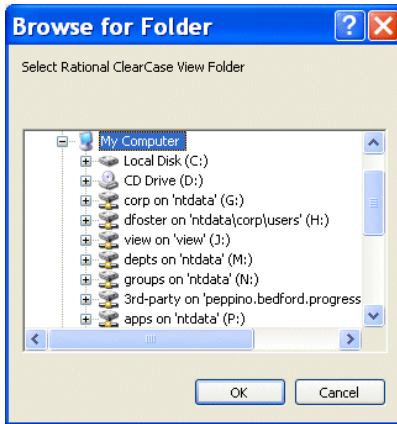



Figure 96. Browse for Folder Dialog Box

7. Navigate to and select the file or directory you want to operate on, or one of the files or directories in the topmost level of the directory hierarchy that you want to access, and click the **OK** button.

Stylus Studio creates a new project that contains the file you selected, or all files that are editable by Stylus Studio and that were in the directory hierarchy of the file you selected. The default name of the project is *Projectn*. To rename the project, select **Project > Save Project As** from the Stylus Studio menu bar.

Adding Files After the Project is Created

After you create the project, you can add additional ClearCase files to it. If the file is already in ClearCase, it must be a sibling of the original file you selected, or it must be a descendant of one of its siblings. If the file you want to add is not in the directory hierarchy of the original file, you must create a new Stylus Studio project and specify a directory in the source control hierarchy that contains all the files you want to be in your Stylus Studio project.

If you want to add a file that is not already in ClearCase, open the file in Stylus Studio and then click **Add To Source Control**  in the Stylus Studio tool bar.

Using Stylus Studio with Zeus CVS

Stylus Studio supports the latest version of the Zeus CVS Provider, and with some additional configuration needed in the **SourceControl > Properties** dialog box.

◆ **To use Stylus Studio to operate on files that are under Zeus CVS source control:**

1. From the Stylus Studio menu bar, select **Project > New Project Wizard**.
The **Project Wizards** dialog box appears.
2. Click **Project from SCC**, and click the **OK** button.
The **Build Project From SCC** dialog box appears.
3. Select **Zeus SCC-CVS** from the **Provider to use** drop-down list.
4. Click the check box for **Recursively import all subprojects**.
5. Click **Advanced**. Several new fields appear.
6. In the **User Name** field, type the user name you want to use to log in to the CVS server.
7. In the **Project Name** field, type the name of a module in the source control hierarchy. This should be the name of a directory that contains all files that you want to open in Stylus Studio.
8. In the **Auxiliary Path** field, type the contents of the CVSROOT environment variable that you use to access the CVS server.

For example, suppose you are required to enter the following commands in a DOS console or UNIX shell:

```
cvs.exe -d:pserver:user@server.company.com:/cvsroot/projectname login
```

```
Password: *****
```

```
cvs.exe -d:pserver:user@server.company.com:/cvsroot/projectname co module
```

The value you should enter in the **Auxiliary Path** field would be:

```
:pserver:user@server.company.com:/cvsroot/projectname
```

9. In the **Working Dir** field, type the name of a local directory.

10. Click the **OK** button.

Stylus Studio downloads the selected files and places them in the directory you specified in the **Working Dir** field. If you move these files from this directory, you must specify the new directory that contains the files in the **Local Project Path** field of the **Source Control Properties** dialog box. To open this dialog box, select **SourceControl > Source Control Properties** from the Stylus Studio menu bar.

All files that can be opened in Stylus Studio are now in the new Stylus Studio project. The default name of the project is *Projectn*. To rename the project, select **File > Project > Save Project As** from the Stylus Studio menu bar.

Note The `cvs.exe` file must be in your `PATH` environment variable.

Specifying Advanced Source Control Properties

The **Advanced** button in the **Build Project From SCC** dialog box displays several additional fields.

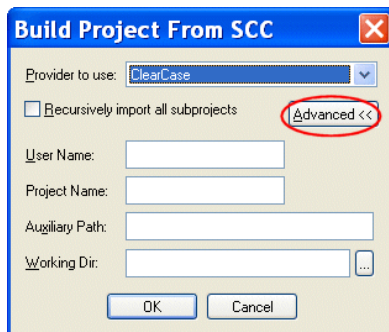


Figure 97. Advanced Source Control Settings

- **User Name** is the name of the source control user. Stylus Studio uses this name to establish a connection with the source control server.
- **Project Name** is the name of the source control repository you want to access. The syntax of the project name depends on the source control provider you want to connect with. For example, SourceSafe uses `$/Name/Name`, ClearCase uses the name of the view, and CVS uses the name of the module. Some source control providers change this description to something more suitable to their model. For example, ClearCase changes it to *ClearCase Attache*.

- **Auxiliary Path** contains source control provider-specific information. This field allows you to enter any other information required to find your source control server. For example, if you are using SourceSafe, you would specify the directory of the SourceSafe client here. If you are using CVS, you would specify the contents of the CVSROOT environment variable.
- **Working Dir** is the local directory into which you copied the files under source control that you want to access. It is the local counterpart for the source control repository. For example, suppose you copied the contents of the SourceSafe repository `$/Company/OneProject` to the local directory `c:\work\myproject`. Your local files would map to the source control hierarchy as shown in [Table 6](#):

Table 6. Local/Repository File Mappings

<i>Local File</i>	<i>Repository File</i>
<code>c:\work\myproject\documentation.xml</code>	<code>\$/Company/OneProject/documentation.xml</code>
<code>c:\work\myproject\subdir\root.java</code>	<code>\$/Company/OneProject/subdir/root.java</code>
<code>c:\work\anotherproject\root.java</code>	<code>\$/Company/anotherproject/root.java</code>

Customizing Tool Bars

Stylus Studio allows you to customize the appearance, location, and content of tool bars, and even to create tool bars of your own. This section covers the following topics:

- “[Tool Bar Groups](#)” on page 119
- “[Showing/Hiding Tool Bar Groups](#)” on page 120
- “[Changing Tool Bar Appearance](#)” on page 121

Tool Bar Groups

Tool bars are organized by functional group within Stylus Studio (Default, Edit, Source Control, and so on). Customizations available for these groups include

- Show/hide
- Look, feel, and size
- Group position

Tip Tool bars are docking windows – you can drag them anywhere on your desktop.

You control all these customizations from the **Toolbars** tab of the **Customize** dialog box.

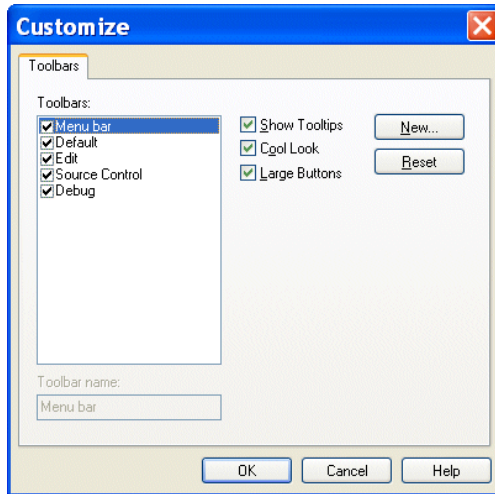


Figure 98. Toolbars Tab of Customize Dialog Box

- ◆ **To display the Customize dialog box, select Tools > Customize from the menu.**

Showing/Hiding Tool Bar Groups

Tool bar groups are displayed by default. Use this procedure to hide/re-display them.

Tip Consider maximizing Stylus Studio on your desktop in order to view as much of the tool bar as possible when making changes.

- ◆ **To hide/show a toolbar group:**
 1. Display the **Customize** dialog box (**Tools > Customize**).
 2. In the **Toolbars** group box, deselect the check box of the group you want to hide. The tool bar is removed from the Stylus Studio window.
 3. To re-display a hidden tool bar group, follow [Step 1](#) and [Step 2](#) and reselect the check box,

Changing Tool Bar Appearance

Changes you can make to the tool bar's appearance include

- Whether or not to show tooltips when the mouse pointer is placed over a tool bar button
- Whether tool bar buttons are rendered in a size larger than the default

Note Appearance settings affect all tool bars. You cannot control the appearance of individual tool bar groups.

◆ **To modify toolbar appearance:**

1. Display the **Customize** dialog box (**Tools > Customize**).
2. Click **Show Tooltips** to toggle the display of tooltips when the pointer is placed over a tool bar button.
3. Click **Large Buttons** to toggle the size of the tool bar buttons.
4. Optionally, click the **Reset** button to restore default settings.
5. Click the **OK** button.

Specifying Stylus Studio Options

Stylus Studio allows you to set a variety of options for Stylus Studio modules, and it provides the ability to define custom tools to run different editors and processors. This section covers the following topics:

- [“Setting Module Options”](#) on page 121
- [“Defining Custom Tools”](#) on page 123

Setting Module Options

Stylus Studio allows you to set a variety of options for the Stylus Studio modules.

◆ **To change module options:**

1. From the Stylus Studio menu bar, select **Tools > Options**.
2. In the **Options** dialog box that appears, expand **Module Settings** to display a list of choices.

XML Diff

You use the **Engine** and **Presentation** pages to define settings used by the XML Diff tool. See “[Diffing Folders and XML Documents](#)” on page 177 for more information.

XML Editor

Click **XML Settings** to specify the following:

- Refresh interval for Sense:X
- Number of errors after which you want Stylus Studio to stop validation, and whether or not you want Stylus Studio to display a message when validation is complete

Click **Custom Validation Engines** to specify an alternate validation engine. See “[Custom XML Validation Engines](#)” on page 1014 for more information.

XSLT Editor

Module settings for the XSLT Editor let you specify external XSLT processors, settings used by the **Mapper** tab, and general editor behavior.

Click **External XSLT** to specify default values for external XSLT processors. Note that Stylus Studio’s back-mapping and debugging features are not supported for all XSLT processors. The XSLT processors that support back-mapping and debugging are identified on the **Processor** tab of the **Scenario Properties** dialog box.

In a scenario, you can specify that you want to use an external XSLT processor. If you use a particular XSLT processor frequently, specify default values here. Then, in the scenario properties, you just need to specify which external XSLT processor you want to use. If you specify default values and you then specify different values in a scenario’s properties, the scenario properties override the defaults. You can specify the following external XSLT options:

- Default custom processor command line
- Default additional path for custom processor
- Default additional classpath for custom processor

Click **Mapper** to specify how `xs1:for-each` instructions should be rendered on the Mapper canvas, and to specify element creation for unlinked nodes. See “[Mapping Source and Target Document Nodes](#)” on page 468 for more information on using the XSLT Mapper.

Click **XSLT Settings** to specify the following:

- Whether Stylus Studio displays the **Scenario Properties** dialog box when you create a new stylesheet
- Whether Stylus Studio saves scenario meta information in stylesheets
- Whether Stylus Studio detects infinite loops
- Maximum recursion level
- Allocated stack size

Java

To modify Java settings, see [“Configuring Java Components”](#) on page 134.


Defining Custom Tools

Stylus Studio allows you to define custom tools to run alternative editors, processors, preprocessors, or postprocessors. For example, you can specify a custom tool that configures Internet Explorer to display the document you are working on.

After you define a custom tool, Stylus Studio adds an entry to its **Tools** menu – select **Tools** and then your tool. The order in which the tool names appear in the **Custom Tools** options page is the order in which the tool names appear in the Stylus Studio **Tools** menu.

◆ To define a custom tool:

1. From the Stylus Studio menu bar, select **Tools > Options**.
Stylus Studio displays the **Options** dialog box.
2. Click **Custom Tools** to display the **Custom Tools** page.

3. In the **Custom Tools** page, click **Define New Tool** .
Stylus Studio displays an entry field for the tool name.

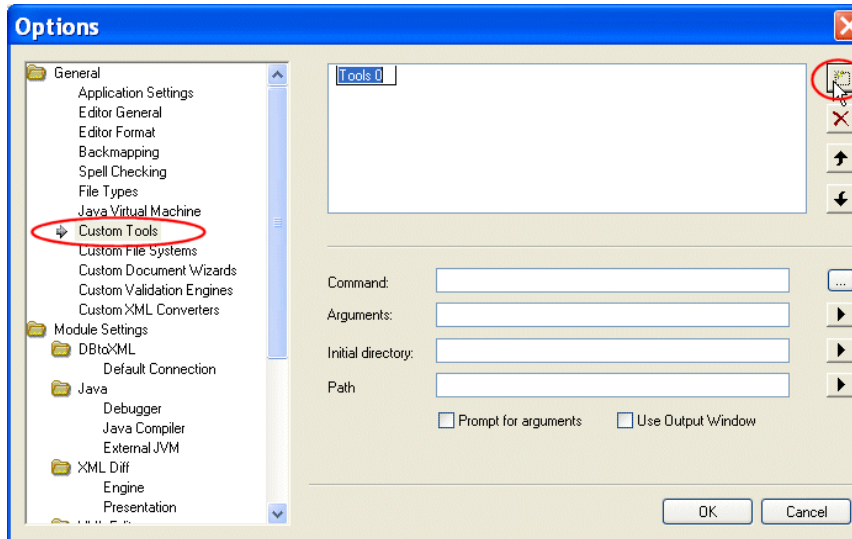



Figure 99. Defining a Custom Tool

4. Enter the name as you want it to appear in the Stylus Studio **Tools** menu.
5. In the **Command** field, specify or select the absolute path for the command that runs your tool. This must be a `.exe`, `.bat`, or `.cmd` file.
6. In the **Arguments** field, specify any arguments your tool requires. You can click  to display a drop-down list that includes **File Path**, **File Dir**, **File Name**, **File Extension**, and **Classpath**.
7. In the **Initial Directory** field, type the absolute path for the directory that contains any files or directories needed by your custom tool.
8. In the **Path** field, type any paths that need to be defined and that are not already defined in your PATH environment variable.
9. If you want Stylus Studio to prompt for arguments before it runs your tool, click **Prompt for Arguments**.
10. If you want Stylus Studio to display output from your custom tool in its **Output Window**, select **Use Output Window**.
11. Click the **OK** button.

Defining Keyboard Shortcuts

You can define a keyboard shortcut for many of the tasks you perform with Stylus Studio. If you find that you repeatedly perform the same action, define a shortcut to speed your work. You can use a keyboard shortcut right after you define it.

How to Define a Keyboard Shortcut

◆ **To define a keyboard shortcut for a Stylus Studio task:**

1. From the Stylus Studio menu bar, select **Tools > Keyboard**.
The **Shortcut Keys** dialog box appears.

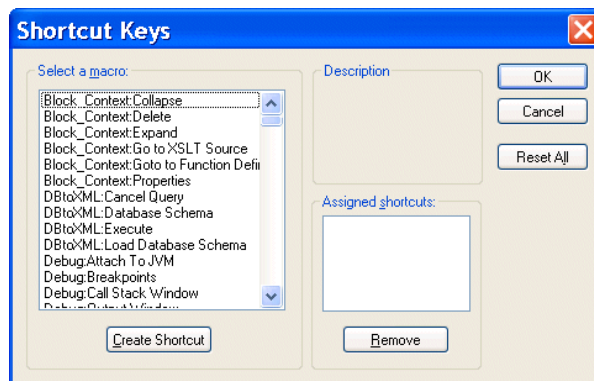


Figure 100. Defining Shortcut Keys

2. In the **Select a macro:** field, select the macro for which you want to define a shortcut.

Tip

When you select a macro, Stylus Studio displays a description of what that macro does.

3. Click **Create Shortcut**.

The **Assign Shortcut** dialog box appears.

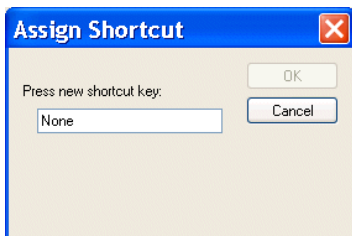


Figure 101. Assigning a Shortcut Key

4. Press the key or keys that you want to be the shortcut. For example, Ctrl+E, F7, Alt+P. The **Assign Shortcut** dialog box displays a message indicating whether or not that key combination is currently in use.
5. If the shortcut key is not already in use, click the **OK** button. Otherwise, try another shortcut key.
Stylus Studio closes the **Assign Shortcut** dialog box.
6. Click **OK** in the **Shortcut Keys** dialog box.

Deleting a Keyboard Shortcut

◆ **To delete a shortcut:**

1. From the Stylus Studio menu bar, select **Tools > Keyboard**. The **Shortcut Keys** dialog box appears.
2. In the **Select a macro:** field, select the macro you want to delete a shortcut for.
3. In the **Assigned shortcuts** field, click the shortcut you want to remove.
4. Click **Remove**.
5. Click **OK** in the **Shortcut Keys** dialog box.

Using Stylus Studio from the Command Line

Stylus Studio provides utilities that allow you to perform several Stylus Studio operations from the command line. These utilities, and where to find more information on them, are described in the following table.

Table 7. Stylus Studio Command Line Utilities

<i>Utility Name</i>	<i>Description</i>	<i>Where to Find More Information</i>
struzzo	Invokes Stylus Studio	“Invoking Stylus Studio from the Command Line” on page 127
StylusDiff	Differs two XML documents	“Running the Diff Tool from the Command Line” on page 205
StylusValidator	Validates XML	“Validating XML from the Command Line” on page 130
StylusXql	Executes an XQuery	“Executing an XQuery from the Command Line” on page 129
StylusXslt	Applies a stylesheet	“Applying a Stylesheet from the Command Line” on page 128

The executables for these command line utilities are located in the `\bin` directory where you installed Stylus Studio.

You can also execute DataDirect XML Converters™ (componentst that let you convert non-XML like EDI and CSV to XML and vice versa) from the command line. To learn more about the DataDirect XML Converters for Java and .NET, see the DataDirect XML Converters documentation at <http://www.xmlconverters.com/doc/>.

Invoking Stylus Studio from the Command Line

You use the `Struzzo` utility to invoke Stylus Studio from the command line and open a particular file. Stylus Studio recognizes the file extension and opens the file in the editor associated with that file type. If Stylus Studio is already running, the same instance is used to open the file specified in the `file` parameter.

You can optionally use the `stylesheet` or `XQuery` parameter to create a scenario with the stylesheet or XQuery you specify.

The Struzzo utility takes the following format:

Struzzo *file* [*stylesheet or XQuery*]

Table 8 describes the parameters for the Struzzo command.

Table 8. Struzzo Command Line Parameters

Parameter	Description
<i>file</i>	The path of the document you want to open in Stylus Studio. This document is used as the source document in a scenario when you provide the <i>stylesheet or XQuery</i> parameter.
[<i>stylesheet or XQuery</i>]	The path of the stylesheet or XQuery you want to use to create a scenario.

Applying a Stylesheet from the Command Line

You use the StylusXslt utility to apply a stylesheet from the command line. The StylusXslt utility uses the XSLT and XPath processors specified in Stylus Studio.

One way you might want to use the StylusXslt command-line utility is to chain stylesheets. That is, you can create a batch file in which Stylus Studio consecutively applies multiple stylesheets to the same XML source document. Stylus Studio creates temporary files to specify the result of one transformation as the source for the next transformation.

The StylusXslt utility takes the following format:

StylusXslt [-out <output file>] [-param name=value] [-fop] [-print] -in <input XML file> <XSLT stylesheet>

Table 9 describes the parameters for the StylusXq1 command.

Table 9. StylusXslt Command Line Parameters

Parameter	Description
[-out <output file>]	File to which the XSLT result will be written. The default is stdout.
[-param name=value]	The name-value pair of a parameter in the stylesheet specified in the <i>stylesheet</i> parameter.
[-fop]	Invokes the Apache Formatting Objects Processor (FOP) to post-process the XSLT result.

Table 9. StylusXslt Command Line Parameters

Parameter	Description
[-print]	Sends the result of the transformation to the default printer.
[-in <input file>]	The path of the XML document to which you want to apply the stylesheet specified in the <i>XSLT stylesheet</i> parameter.
<XSLT stylesheet>	The path of the XSLT you want to apply to the document specified in the -in parameter.

Executing an XQuery from the Command Line



XQuery support is available only in Stylus Studio XML Enterprise Suite and Stylus Studio XML Professional Suite.

You use the StylusXq1 utility to execute an XQuery from the command line. The format for invoking the utility is as follows:

```
StylusXq1 [-in <source>] [-out<output file>] [-param name+value] [-i] [-debug host[:port]] <XQuery file>
```

Table 10 describes the parameters for the StylusXq1 command. All parameters are required.

Table 10. StylusXq1 Command Line Parameters

Parameter	Description
-in <source>	The path of the XML document to be used to set the current context for the XQuery.
-out <output file>	File to which you want the XQuery result to be written. The default is stdout.
-param name=value	The name-expression pair of a variable in the XQuery specified in the <i>XQuery file</i> parameter.
-i	Indents the XQuery result.

Table 10. StylusXql Command Line Parameters

Parameter	Description
<code>-debug host[:port]</code>	Debugs the query using the debug server specified by the host and, optionally, port parameters.
<code>XQuery file</code>	The path of the XQuery you want to execute against the file specified in the <code>-in <source></code> parameter.

Validating XML from the Command Line

You use the `StylusValidator` utility to validate XML from the command line. `StylusValidator` uses the built-in Stylus Studio XML validator. All output from this utility goes to `stdout`.

The `StylusValidator` utility takes the following format:

```
StylusValidator [-q] [-noval] [-schema] filename
```

[Table 11](#) describes the parameters for the `StylusXql` command.

Table 11. StylusValidator Command Line Parameters

Parameter	Description
<code>[-q]</code>	Quiet mode – errors are not printed to <code>stdout</code> .
<code>[-noval]</code>	Checks only for well-formedness. Does not check for errors.
<code>[-schema file]</code>	Validates the XML document against the XML Schema specified in the <code>file</code> parameter.
<code>filename</code>	The path of the XML document you want to validate.

Managing Stylus Studio Performance

Stylus Studio uses the `TEMP` directory to store temporary files such as the translation in UNICODE of the current XML or XSLT document. File systems are usually quite fast when handling files that are in the range of a few hundred megabytes. Stylus Studio performance should be smooth and quick when the `TEMP` windows environment variable points to a location where

- There is a minimum of 1 gigabyte of free space.

- The host disk is reasonably fast.

Stylus Studio is regularly tested against files that are up to 120 MB. How well your installation of Stylus Studio can create, open, and manipulate such large files, or even larger files, depends on

- Available physical memory
- Dimension of the page file
- Current load of the machine

Troubleshooting Performance

Table 12, [Performance Symptoms](#), summarizes performance symptoms you might experience and where to find information on addressing them.

Table 12. Performance Symptoms

<i>Symptom</i>	<i>See</i>
XML editing is slow	Changing the Schema Refresh Interval on page 131 Checking for Modified Files on page 132
Errors or crashes during XSLT processing	Changing the Recursion Level or Allocated Stack Size on page 133
Stylus Studio is slow to start	Automatically Opening the Last Open Files on page 133

Changing the Schema Refresh Interval

As you edit an XML document, Stylus Studio displays a pop-up menu that lists the elements and element attributes you can create. Stylus Studio retrieves this information from the document's schema. The frequency with which Stylus Studio retrieves this information can affect XML editing performance. The default refresh interval is 10 seconds.

If XML editing performance is slow, increase the refresh interval that Stylus Studio uses to refresh the schema information.

◆ To change the refresh interval:

1. From the Stylus Studio menu bar, select **Tools > Options**.
The **Options** dialog box appears.

2. Click **Module Settings > XML Editor > XML Settings**.

The **XML Settings** page of the **Options** dialog box appears.

3. In the **Refresh interval** field, type a larger number. For information about how Stylus Studio uses this interval, see [“Options - Module Settings - XML Editor - XML Settings”](#) on page 1155.

Tip

If the schema used by your document is almost never modified, you can safely increase the interval to as much as 10,000 seconds.

4. Click **OK**.

Checking for Modified Files

When you are working with files that Stylus Studio must open through network connections that might be slow, you might not want Stylus Studio to automatically check for modified files. Turning off this option can improve XML editing performance.

◆ **To turn off checking for modified files:**

1. From the Stylus Studio menu bar, select **Tools > Options**.

The **Options** dialog box appears.

2. Click **Application Settings** if it is not already selected.

The **Application Settings** page of the **Options** dialog box appears.

3. If the **Automatically check for externally modified files** is selected, deselect it. For information about how Stylus Studio uses this setting see [“Options - Application Settings”](#) on page 1129.

Alternatively, you can select **Disable check on hidden files**, which allows Stylus Studio to skip these files. Hidden files are files that are in the Stylus Studio project or the **Other Documents** folder but are not currently open in Stylus Studio.

4. Click **OK**.

Changing the Recursion Level or Allocated Stack Size

If you are getting errors or crashes when you use the internal Stylus Studio XSLT processor, there are two options you can change to fix this.

- The **Maximum recursion level** is the number of levels Stylus Studio allows you to recurse on a template invocation.
- The **Allocated stack size** is the amount of memory allocated to the XSLT processing thread stack.

◆ **To change the recursion level or the allocated stack size:**

1. From the Stylus Studio menu bar, select **Tools > Options**.
The **Options** dialog box appears.
2. Click **Module Settings > XSLT Editor > XSLT Settings**.
The **XSLT Settings** page of the **Options** dialog box appears.
3. Adjust the **Maximum recursion level** and the **Allocated stack size** as needed. For information about how Stylus Studio uses these settings see [“Options - Module Settings - XSLT Editor - XSLT Settings”](#) on page 1164.
4. Click **OK**.

Automatically Opening the Last Open Files

When you start Stylus Studio, it automatically opens any files that were open the last time you closed it. This feature can affect performance if many files were open when you last closed Stylus Studio.

If Stylus Studio is taking a long time to start, you can do one of the following:

- Close most or all files before you shut down Stylus Studio.
- Turn off the option that automatically opens the files that were open the last time you closed Stylus Studio.

◆ **To prevent Stylus Studio from automatically opening documents:**

1. From the Stylus Studio menu bar, select **Tools > Options**.
The **Options** dialog box appears.
2. Click **Application Settings** if it is not already selected.
The **Application Settings** page of the **Options** dialog box appears.

3. If **Open last documents automatically when Stylus Studio starts** is selected, deselect it. For information about how Stylus Studio uses this settings see [“Options - Application Settings”](#) on page 1129.
4. Click **OK**.

Configuring Java Components

Several modules in Stylus Studio are written using Java, and therefore require either Java runtime or Java compiler components to be installed on your computer. These Java components, the Java Runtime Environment (JRE) and the Java Development Kit (JDK), are available from Sun Microsystems and are installed separately from Stylus Studio.

You can install these components either before or after you install Stylus Studio. When you start Stylus Studio, it attempts to identify the location of the Java runtime libraries and compiler automatically.

This section identifies the Stylus Studio modules that require Java runtime and Java compiler components, where you can download these Java components, and how to force Stylus Studio to detect new or changed Java components.

This section covers the following topics:

- [“Stylus Studio Modules That Require Java”](#) on page 134
- [“Verifying the Current Java Virtual Machine”](#) on page 135
- [“Downloading Java Components”](#) on page 135
- [“Modifying Java Component Settings”](#) on page 136

Stylus Studio Modules That Require Java

The following modules in Stylus Studio require that Java runtime and/or Java compiler components are installed on the machine on which you have installed Stylus Studio:

- Saxon XSLT and XQuery engines
- Built-in Java XSLT processor
- FOP
- Custom File Systems
- Web Service Call Composer Axis client
- DataDirect XML Converters accessed via URL
- Sense:X in the Java editor

Rather than trying to determine in advance which Stylus Studio modules you might use in your XML application development, consider installing the JDK or JRE on your machine.

Settings for Java Debugging

See “[Debugging Java Files](#)” on page 503 for more information on this topic.

Verifying the Current Java Virtual Machine

The Java Virtual Machine (JVM) interprets runtime commands and compiler instructions; it is part of the Java installation. You can check to see the current version of the JVM installation by selecting **Help > About** from the Stylus Studio menu:

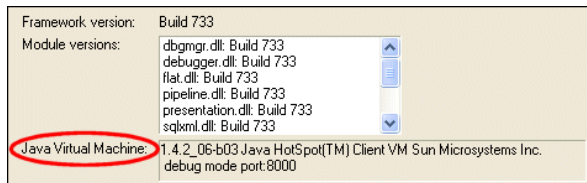


Figure 102. Verifying the Current JVM Installation

The **Java Virtual Machine** field displays information about the JVM installed on your machine.

Downloading Java Components

Java runtime and compiler components are available for download from Sun Microsystems; they are packaged in the Java 2 Platform Standard Edition (J2SE).

Either of the following versions is compatible with Stylus Studio 2007:

- J2SE 1.4.2 (download it here: <http://www.java.sun.com/j2se/1.4.2/download.html>)
- J2SE 5.0 (download it here: <http://www.java.sun.com/j2se/1.5.0/download.jsp>)

Modifying Java Component Settings

Properties for JVM and JDK components are displayed on the **Java Virtual Machine** page of the **Options** dialog box, shown in **Figure 103**. When you start Stylus Studio, it automatically detects the Java Virtual Machine (JVM) and compiler components installed on your machine and sets the properties for these components accordingly.

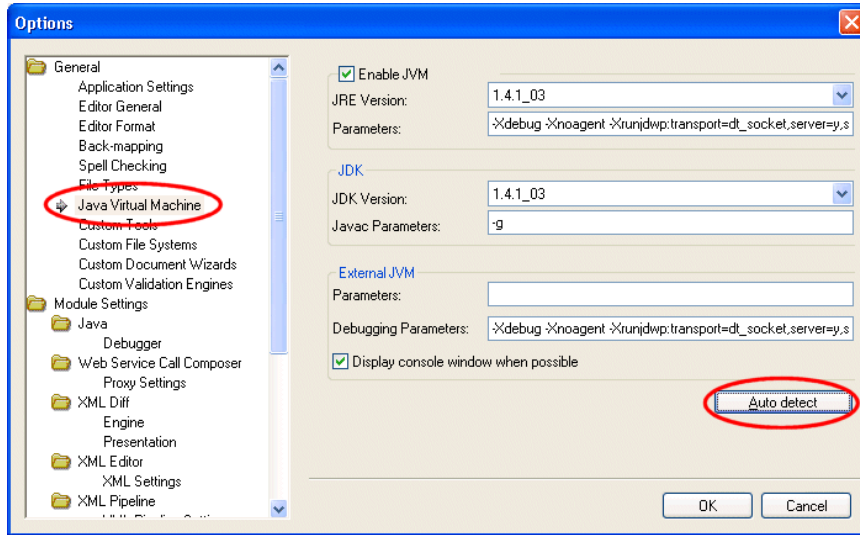


Figure 103. Reset Java Properties in the Options Dialog Box

Once these properties have values, Stylus Studio uses them until you either

- Use the auto detect feature to change them. You might want use auto detect if you have been using Stylus Studio with the J2SE 1.4.2 and later install the J2SE 5.0, for example.
- Change them manually. You can manually specify that Stylus Studio use a different jvm.dll or javac.exe, for example.

How Auto Detect Works

The auto detect feature prompts Stylus Studio to fetch the settings from the registry setting **Current Version** under the key **HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Java Runtime Environment** to find the version, and then adds the version number to that same location to get the settings.

Note that if you manually change your settings to use another local version of the JDK, it may fail to load properly unless you also point the `Current Version` setting to match. This is because the JVM itself might try to load DLLs from the location of the current version instead of the location you specify.

About JVM Parameters

As a rule, you should never change the default values in the **Parameter** fields for the **JVM** or the **External JVM**. This option exists to accommodate unusual configurations. In such situations, Stylus Studio Technical Support might instruct you to change this value.

About JDK Parameters

The `-g` parameter instructs the compiler to add debugging information to the generated `.class` file; it is set by default.

How to Modify Java Component Properties

◆ To modify Java component properties:

1. Start Stylus Studio if it is not already running, and select **Tools > Options** from the menu.
The **Options** dialog box appears.
2. Select **General > Application Settings > Java Virtual Machine**.
3. If you want Stylus Studio to update Java component properties to the latest installed version on your machine, click the **Auto detect** button.
Otherwise, make the changes manually.
4. Click **OK**.
5. If you made changes to any JVM properties, you need to restart Stylus Studio for those changes to take effect.

Chapter 2 **Editing and Querying XML**

Stylus Studio makes it easy to create, edit, and query XML documents. Depending on the structure of the data in your XML document, you can choose to work with raw XML text, a DOM tree diagram, or a grid representation. Any changes you make in one view are immediately visible in every other view. You can easily create an XML Schema or DTD based on the content of your XML document if it is not already associated with a schema.



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the XML Editor video](#).

You can see other Stylus Studio video demonstrations here:
http://www.stylusstudio.com/xml_videos.html.

This section covers the following topics:

- [“Creating XML Documents”](#) on page 140
- [“Using Document Wizards to Create XML”](#) on page 141
- [“Updating XML Documents”](#) on page 143
- [“Using the Text Editor”](#) on page 145
- [“Updating DOM Tree Structures”](#) on page 158
- [“Using the Grid Tab”](#) on page 162
- [“Diffing Folders and XML Documents”](#) on page 177
- [“Using Schemas with XML Documents”](#) on page 208
- [“Converting XML to Its Canonical Form”](#) on page 211
- [“Querying XML Documents Using XPath”](#) on page 211
- [“Printing XML Documents”](#) on page 211
- [“Saving XML Documents”](#) on page 212

Creating XML Documents

You can create XML documents in Stylus Studio manually, using the XML Editor, or automatically, using document wizards, the Stylus Studio Custom XML Conversions module, DataDirect XML Converters, and other features.

Using the XML Editor

To create an XML document using the XML Editor, select **File > New > XML Document** from the Stylus Studio menu.

Stylus Studio displays a new, untitled document in the XML Editor. The document contains only the XML declaration:

```
<?xml version="1.0"?>
```

The XML Editor provides several views of an XML document, each on its own tab – **Text**, **Tree**, **Grid**, and **Schema**. See [“Updating an XML Document – Getting Started”](#) on page 6 for an overview of these XML editing tools. See [“Updating XML Documents”](#) on page 143 for more detailed information.

Other Ways to Create XML

You can also create XML using

- Document wizards that convert HTML, DTD, and XML Schema to XML. See [“Using Document Wizards to Create XML”](#) on page 141, later in this section.
- DataDirect XML Converters that convert CSV, fixed-width, EDI, and other flat file formats to XML. See [“DataDirect XML Converters”](#) on page 217.
- User-defined converters that you build using Stylus Studio’s Custom XML Conversions module. See [“Custom XML Converters”](#) on page 223.



DataDirect XML Converters and the Stylus Studio Custom XML Conversions module are available only in Stylus Studio XML Enterprise Suite.

Using Document Wizards to Create XML

Stylus Studio provides several document wizards that automatically create XML documents from XML Schema, DTD, and HTML. This section describes how to work with these document wizards; it covers the following topics:

- “How to Use a Document Wizard” on page 141
- “Creating XML from XML Schema” on page 141
- “Creating XML from DTD” on page 142
- “Creating XML from HTML” on page 142

How to Use a Document Wizard

Most document wizards operate in the same general fashion:

1. Select the document wizard you want to use.
2. Specify the file from which you want to create an XML document (an HTML file or an XML Schema, for example).
3. Specify additional settings that will affect the resulting XML document (the root node, for example).
4. Run the wizard.

Converted files are opened as new, untitled XML documents in the XML Editor.

All document wizards are listed in the **Document Wizards** dialog box. Select **File > Document Wizards** to display this dialog box.

Creating XML from XML Schema

When you use the XML Schema to XML document wizard, you specify the XML Schema from which you want to create an XML document, as well as its root node, and whether or not you want to generate comments in the XML.

Note If the XML Schema contains an element defined using a built-in type, the instance of that element in the XML document is created using the minimum value of the range specified for that type. For example, if the XML Schema contains a `<part>` element defined as `type="xs:integer"`, the `<part>` element in the resulting XML document appears as `<part>-9223372036854775808</part>`.

Creating XML from DTD

In addition to XML Schema, you can use a DTD to create an XML file. When you use the DTD to XML document wizard, you specify the DTD from which you want to create an XML document in the **DTD File** field. Next, specify the element that you want to be the root element in the new XML document, and whether or not you want expand each element only once (this results in a smaller XML document file).

Creating XML from HTML

You can create an XML document from an HTML file using the HTML to XML document wizard. Simply specify the HTML file you want converted to XML and run the document wizard.

Tip Stylus Studio also has a document wizard that converts HTML to XSLT. See [Creating a Stylesheet from HTML](#) on page 362.

Updating XML Documents



The XML editor **Grid** tab is available only in Stylus Studio XML Enterprise Suite and Stylus Studio XML Professional Suite.

Stylus Studio provides **Text**, **Tree**, and **Grid** views for updating any XML document you open. The view you choose for editing depends on how structured your data is and your personal preferences. This section describes how to choose an XML document view to work with and other features related to editing XML.

This section discusses the following topics:

- “[Choosing a View](#)” on page 143
- “[Saving Your Work](#)” on page 144
- “[Ensuring Well-Formedness](#)” on page 144
- “[Reverting to Saved Version](#)” on page 144
- “[Updating Java Server Pages as XML Documents](#)” on page 145

Choosing a View

You can add and modify the data and structure of an XML document in any view. When you switch to a different view, any changes you made appear in the new view. To move from view to view, click the **Text**, **Tree**, or **Grid** tab at the bottom of the document you are working with.

To add contents to an empty XML document, consider the structure of the data you plan to add. The **Grid** view is most useful for creating very structured data that includes multiple instances of the same elements. The **Tree** view makes it easy to add many different elements. In order to use it, however, the XML must be well-formed.

Each view of the document allows you to query the contents of the document. See “[Querying XML Documents Using XPath](#)” on page 211.



Watch it! You can view a video demonstration of the **Grid** tab by clicking the television icon or by clicking this link: [watch the XML Grid Editor video](#).


A complete list of the videos demonstrating Stylus Studio’s features is here: http://www.stylusstudio.com/xml_videos.html.

For More Information

To learn more about a specific XML view, see one of the following sections:

- “Using the Text Editor” on page 145
- “Updating DOM Tree Structures” on page 158
- “Using the Grid Tab” on page 162

Saving Your Work

The procedure for saving your work is the same regardless of which view you use to edit XML – make sure your work is in the active window, and then select **File > Save** from the Stylus Studio menu bar, or click **Save**  in the Stylus Studio tool bar.


Ensuring Well-Formedness

- ◆ **To ensure that your XML document is well formed, click the **Tree** tab at the bottom of the XML editor window.**

If the document is well formed, Stylus Studio displays the tree representation. If the document is not well formed, Stylus Studio displays a message that indicates the reason the document is not well formed and the location of the error or omission. Correct the document, and click the **Tree** tab.

If you are already viewing the **Tree** representation of your document, the document is well formed. When you edit the **Tree** view, the XML that Stylus Studio generates is always well formed.

Reverting to Saved Version

You might make some changes to an XML document and then decide that you do not want to save them. In the Stylus Studio tool bar, click **Reload** . Stylus Studio displays a message that warns you that you will lose any changes, and prompts you to confirm that you want to reload the version of the document that is in the file system. After you confirm, Stylus Studio displays the last saved version of the document.

Updating Java Server Pages as XML Documents

◆ **To open a .jsp file as an XML document:**

1. In the File Explorer, navigate to the JSP file you want to open.
2. Right click the file name, and select **Open With** from the shortcut menu.
3. Click **XML Editor**.

Using the Text Editor

You use the **Text** tab of the XML editor to edit XML text. The **Text** tab provides the usual tools you expect to find in a text editor. These tools are described in this section.

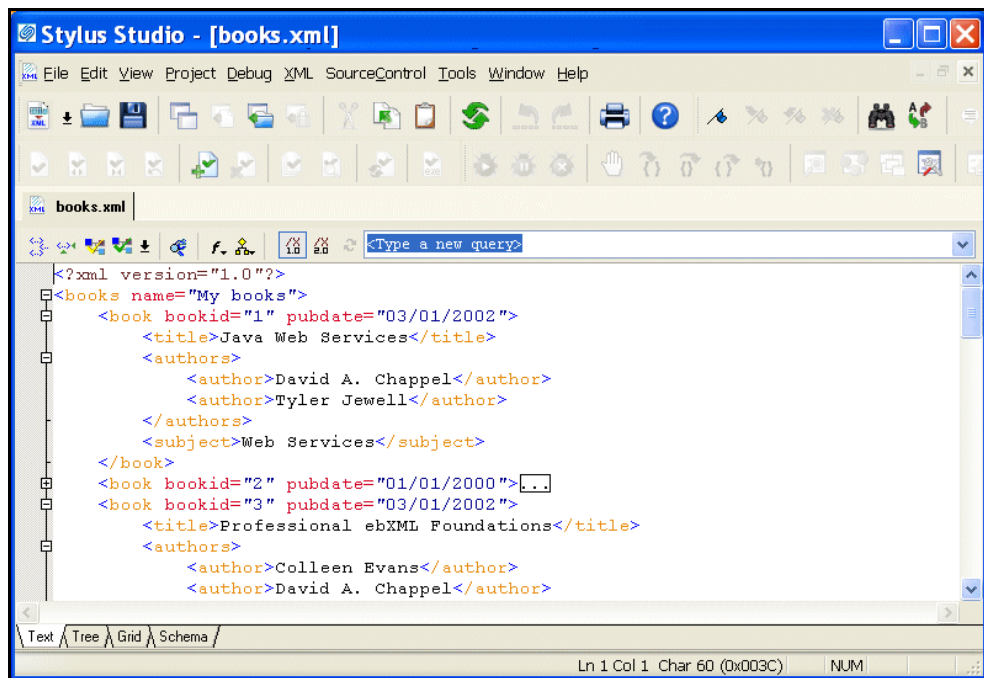


Figure 104. Text Tab in the XML Editor

This section covers the following topics:

- [Text Editing Features](#) on page 146
- [Use of Colors in the Text Tab](#) on page 151

- [Using the Spell Checker](#) on page 152
- [Moving Around in XML Documents](#) on page 156

Text Editing Features

This section describes some of the more common text editing tools and features in Stylus Studio.

Simple Text Editing

Select the text you want to edit and then do any of the following:

- Click the right mouse button to display a pop-up menu of edit commands.
- Click the appropriate button in the Stylus Studio tool bar.
- Press the standard control keys to copy, cut, paste, undo, or redo.

You can select a portion of text and move it to a new location by dragging it. You can drag text from one document to another. You can drag text from documents outside Stylus Studio to a document in Stylus Studio.

Code Folding

Code folding is the ability to collapse three or more lines of code in XML-based editors. For example, this code segment:

```
<author>
    <first-name>Joe</first-name>
    <last-name>Bob</last-name>
    <award>Trenton Literary Review Honorable Mention</award>
</author>
```

when collapsed, appears as this:

```
<author>
```

Code folding allows you to simplify the visual presentation of XML-based code; folding does not affect the underlying code.

In editors in which code folding is supported, Stylus Studio displays a tree control in the gutter to the left of editing canvas; this is the same area of the editor used to display [line numbers](#), debugging symbols, and back mapping symbols. By default, all code is displayed.

What You Can Fold

In XML-based editors, you can fold

- Internal DTD
- Comments
- CDATA
- XML elements

In the XQuery Source editor, you can fold

- Comments
- Expressions delimited by curly braces ({ and })
- XML elements

How to Fold Code

- ◆ **To unfold a segment of code, click the [-] symbol associated with that code.**

When you fold code, Stylus Studio displays a boxed ellipsis symbol at the end of the line of code you have folded, as shown here:

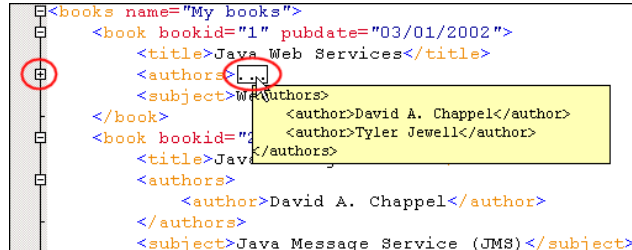


Figure 105. Example of Folded Code

If you place the pointer in the ellipsis symbol, Stylus Studio displays a tool tip that shows you the collapsed code. The amount of code that appears in the tool tip depends on the area on your desktop you have given the Stylus Studio application.


Tip You can unfold a folded segment by double-clicking the tool tip.

Sense:X

As you type, Sense:X prompts you with the possible tags that you can insert at a given location based on the XML Schema associated with the document you are editing. As soon as you type a tag's open bracket, Stylus Studio displays a scrollable list of the

elements that are allowed at that location of the document. Double-click the tag you want. For example, suppose you have an XML Schema where the book element can contain author or title elements. If you add a book element, the Sense:X list displays only the author and title tags.


Indent

Indent XML tags to show the hierarchy relationships. Click **Indent XML Tags** . Stylus Studio indents all text in the active XML document window.

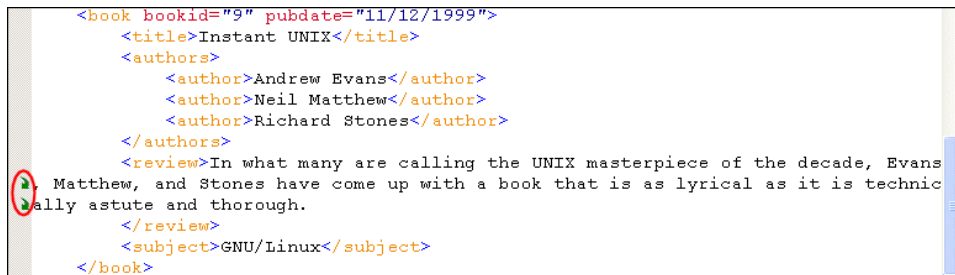
Note After you click the **Indent XML Tags** button, you cannot automatically undo or redo any changes you have been making. After you make more changes, you can press Ctrl+Z and Ctrl+Y to automatically undo and redo those changes until you click **Indent XML tags** again.

Line Wrap

Stylus Studio automatically wraps lines whose length exceeds 16k characters. You can turn off this feature by selecting **Disable** from the **Line wrap** field on the **Editor General** page of the **Options** dialog box (**Tools > Options**).

You can override line wrap settings by selecting **Edit > Wrap Lines** from the Stylus Studio menu or by clicking the wrap lines button on the tool bar(.

When line wrapping is on, Stylus Studio wraps lines to fit in the available window; the place at which the line wraps moves as the width of the window changes. Green arrows, as shown in [Figure 106](#), identify lines that have wrapped.



```
<book bookid="9" pubdate="11/12/1999">
  <title>Instant UNIX</title>
  <authors>
    <author>Andrew Evans</author>
    <author>Neil Matthew</author>
    <author>Richard Stones</author>
  </authors>
  <review>In what many are calling the UNIX masterpiece of the decade, Evans
  Matthew, and Stones have come up with a book that is as lyrical as it is technic
  ally astute and thorough.
  </review>
  <subject>GNU/Linux</subject>
</book>
```

Figure 106. Green Arrows Identify Lines That Have Wrapped

Spell Checking


By default, Stylus Studio spell checks text as you type using an internal spell checker. Words the spell checker believes are misspelled (or repeated) are underlined with a squiggly line, as shown in [Figure 107](#).

```
<book bookid="5" pubdate="11/10/2000">
  <title>Beginner&apos;s Guide to Access 2.0</title>
  <authors>
    <author>Worx Author Team</author>
    <author>minollo@minollo.com</author>
  </authors>
  <subject class="1">Access</subject>
</book>
```


Figure 107. Typographical Errors Are Highlighted by the Spell Checker

For more information, see [Using the Spell Checker](#) on page 152.

Font

You can change the font of the text display in Stylus Studio. This change affects only the Stylus Studio display. Beyond personal preference, you might choose to change the font for localization purposes – the available fonts are the fonts that can display the characters in your XML file. For example, in a Japanese file, only two or three font names appear. Click **Font Change**  to display a list of fonts.

Comments

Select the text that you want to be a comment. In the Stylus Studio tool bar, click **Comment/Uncomment Selection** . To remove comment tags, select the commented text, and click **Comment/Uncomment Selection**.

Tip To select an entire line, click the gray area to the left of the line you want to select.

Bookmarks

You can set bookmarks in the XML display. Bookmarks allow you to jump to important lines in your file. See [“Using Bookmarks”](#) on page 498.

Search/Replace

Search for and replace text you specify. Click **Find**  or **Replace**  in the tool bar. You can also enable Find by pressing Ctrl + F.

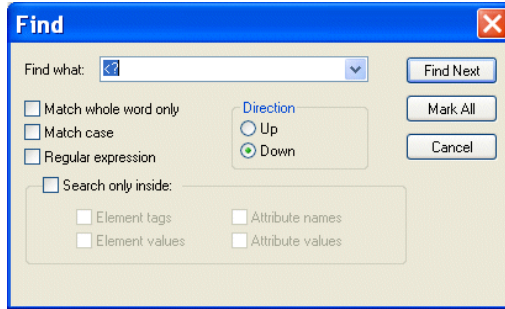


Figure 108. Find Dialog Box

When you enable Find, Stylus Studio displays the word in which the cursor is located – whether the cursor is within the word or immediately adjacent to it – in the **Find what** field of the **Find** dialog box. Similarly, any text you have selected – whole, partial, or multiple words – is displayed in the **Find what** field.

Tip You can scroll through a list of the other words you have searched for by clicking the down arrow when the **Find what** field is active.

Note that in addition to specifying case, you can also indicate whether or not you want to use regular expressions in the **Find what** field and **Replace with** field. This allows you, for example, to search for a line and replace it with multiple lines, as shown in the following example.

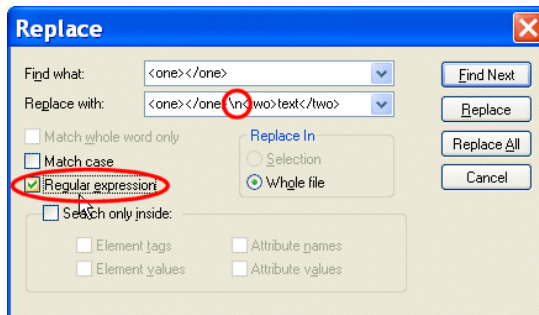


Figure 109. Replacing Text Using Regular Expressions

See “[Sample Regular Expressions](#)” on page 258 for examples of regular expressions and to learn about other sources of information.

Use of Colors in the Text Tab

Stylus Studio text editors use colors to distinguish the types of data in XML documents. The default colors for the XML Editor are described in the [Table 13](#).

Table 13. Text Colors in Stylus Studio

<i>Color</i>	<i>Type of Data</i>
White	Document background
Royal blue	Markup
Black	XML declaration and text node contents
Pale blue	Schema definition
Purple	Element names defined in the DTD
Red	Attribute names
Dark blue	Attribute values
Orange	Element names not defined in the DTD

How to Change Colors

You can set colors for the text editors associated with different document types (XML, XQuery, XSLT, and so on) individually.

◆ To change colors:

1. Select **Tools > Options** to display the **Options** dialog box.
2. Click **Editor Format**.
3. Select the editor type from the **Editor** drop-down list.
4. Set the font, size, and color for different document categories as desired.
5. Click **OK** to close the **Options** dialog box.

Using the Spell Checker

You can use the Stylus Studio Spell Checker with all of Stylus Studio’s text-based editors (like editors for XQuery and XSLT, for example) to both actively and passively check your documents for typographical errors such as misspellings and repeated words.

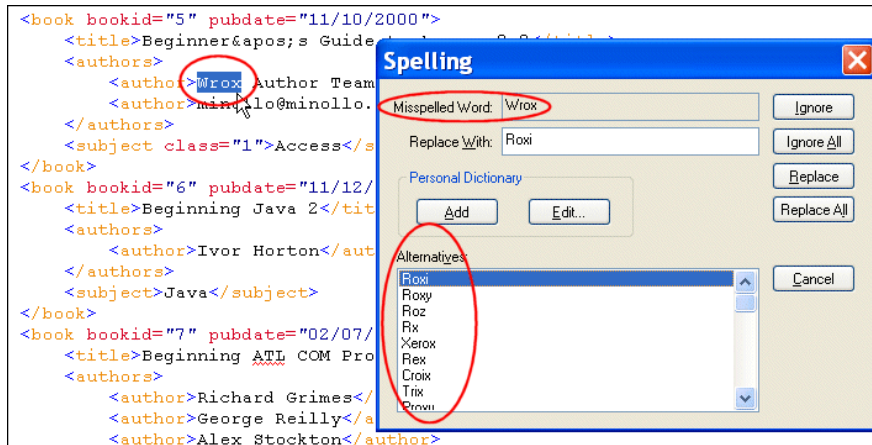


Figure 110. Stylus Studio’s Spell Checker

Default Spell Checking

The Spell Checker is on by default for most editors. This means that when you open a document in a Stylus Studio text editor, and as you type in that document, Stylus Studio checks the document for typographical errors. Words that the Spell Checker identifies as possibly containing a typographical error are underlined with a red “squiggle”, like the word *Wrox* shown in Figure 107.

Tip You can right-click a word with a squiggle and select **Spell Checker Suggestions for** to display a list of suggestions for the word identified by the Spell Checker.

Manual Spell Checking

At any time, you can manually spell check a document by selecting **Tools > Check Spelling** from the Stylus Studio menu. When you do this, Stylus Studio starts the Spell Checker, which reads through the current document. When it finds a possible typographical error, Stylus Studio displays the **Spelling** dialog box, as shown in Figure 110.

Using the **Spelling** dialog box, you can

- Ignore the current occurrence of the word the Spell Checker has selected
- Ignore all occurrences of the word
- Replace the current occurrence of the word
- Replace all occurrences of the word
- Add new words to the dictionary
- Edit existing dictionary content

Specifying Spell Checker Settings

You specify Spell Checker settings using the **Spell Checking** page of the **Options** dialog box.

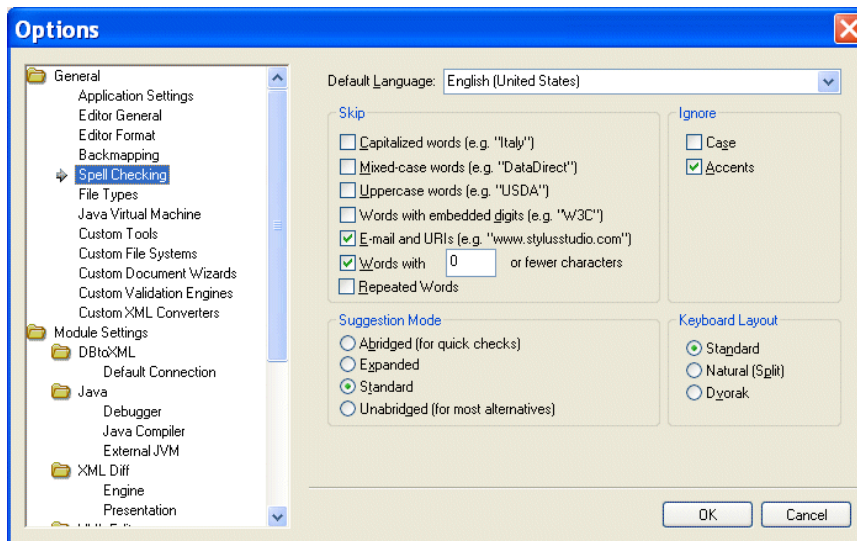


Figure 111. Options for the Spell Checker

Spell Checker settings include

- Words to skip based on certain characteristics – you might decide to skip e-mail addresses and URLs, for example. Skipped words are not considered by the Spell Checker.
- Characteristics in words that you wish to ignore – you might not care about case or accents marks for spelling purposes. In this case, two words that share the same spelling, except for the characteristic you specify, are considered to be equivalent (*même* and *meme* (accent), or *BMW* and *bmw* (case), for example).

- The type of dictionary you want the Spell Checker to use when providing alternatives to the typographical errors it locates. Settings range from **Abrided** to **Unabridged** and show the fewest to the most alternatives, respectively. The **Abrided** setting results in fewer alternative suggestions for misspelled words than **Standard** (the default) or **Unabridged**, for example, but it requires less time to spell check a given document.
- The layout of the keyboard you are using. The Spell Checker uses this information to offer meaningful suggestions to words you might have mistyped.

How to Spell Check a Document

◆ **To spell check document:**

1. Select **Tools > Check Spelling** from the menu.

Stylus Studio starts checking the document for typographical errors. If it finds a typographical error, it displays the **Spelling** dialog box.

<i>If You Want To</i>	<i>Then</i>
Replace the misspelled word with the word suggested by the Spell Checker	Click Replace . Click Replace All to replace all occurrences of that word. You can also replace the misspelled word with another word selected from the Alternatives list box, or with a word you type in the Replace with field.
Ignore the misspelled word	Click Ignore . Click Ignore All to ignore all occurrences of that word.
Add the misspelled word to the personal dictionary	Click Add .
Edit the personal dictionary	Click Edit . See Using the Personal Dictionary on page 155 for more information.

2. Once you select an action, the Spell Checker continues checking the document. When you have addressed all identified errors in the document (either by replacing, correcting, or ignoring them), the Spell Checker stops.

Using the Personal Dictionary

The Stylus Studio Spell Checker comes with its own dictionary. You can create a *personal dictionary* and fill it with your own entries. Personal dictionaries are used in conjunction with the Spell Checker dictionary across all Stylus Studio editors.

To add entries to the personal dictionary, you can

- Type entries individually
- Import lists formatted as .txt files
- Automatically add entries while you check the document

The personal dictionary is stored in the c:\Documents and Settings\username\Application Data\Stylus Studio directory.

Warning Do not modify the files in this directory by hand. Use the **Personal Dictionary Editor** dialog box to make any changes to the personal dictionary.

◆ To add a word to the personal dictionary:

1. Start the Spell Checker and display the **Personal Dictionary Editor** dialog box (click **Edit** on the **Spelling** dialog box).
2. Enter a word in the **New Word** field.
3. Click the **Add** button.
The word appears in the **Words in Personal Dictionary** list box.
4. Click the **Close** button.

Tip You can also add any word identified as a misspelling to the personal dictionary by pressing the **Add** button on the **Spelling** dialog box.

◆ To import lists into the personal dictionary:

Note Lists you import into the personal dictionary must be unformatted .txt files, with each entry on its own line. Do not use tab- or comma-separated files.


1. Start the Spell Checker and display the **Personal Dictionary Editor** dialog box (click **Edit** on the **Spelling** dialog box).
2. Click the **Import** button.
The **Open** dialog box appears.
3. Select the .txt file you want to import into the personal dictionary and click **Open**.
The words in the list you import appear in the **Words in Personal Dictionary** list box.

4. Click the **Close** button.
- ◆ **To export the personal dictionary to a .txt file:**
 1. Start the Spell Checker and display the **Personal Dictionary Editor** dialog box (click **Edit** on the **Spelling** dialog box).
 2. Click the **Export** button.
The **Save As** dialog box appears.
 3. Navigate to the directory in which you want to save the copy of the personal dictionary.
 4. Enter a name in the **File name** field.
 5. Click **Save**.
The contents of the personal dictionary is saved to the text file.
 6. Click the **Close** button.

Moving Around in XML Documents

Stylus Studio provides several tools for easily moving around in an XML document.

Line Numbers




To go to a particular line, click **Go to a specified location**  in the Stylus Studio tool bar. Stylus Studio displays the **Go To** dialog box. Type the number of the line you want to go to and click **OK**. The cursor moves to the line you specified.

Stylus Studio displays line numbers and column numbers in the lower right corner of the Stylus Studio window. If you want, you can set a Stylus Studio option that displays line numbers to the left of each line in the editor you are using.


To do this, from the Stylus Studio menu bar, select **Tools > Options**. In the **Options** page that appears, click **Editor General**. In the **Editor** field, select the editor in which you want to display line numbers. Select the **Show Line Numbers** check box.

Bookmarks


To quickly focus on a particular line, insert a bookmark for that line. You can insert any number of bookmarks. You can insert bookmarks in any document that you can open in Stylus Studio.

To insert a bookmark, click in the line that you want to have a bookmark. Then click **Toggle Bookmark**  in the Stylus Studio tool bar. Stylus Studio inserts a turquoise box with rounded corners to the left of the line that has the bookmark. To move from bookmark to bookmark, click **Next Bookmark**  or **Previous Bookmark** . See “Using Bookmarks” on page 498.

Tags

To move to the closing tag for an element, click in the tag name for the element. In the Stylus Studio tool bar, click **Go to Matching Tag** . Stylus Studio moves the cursor to the closing tag for the element you clicked.

Find

In any view of an XML document, and in the **XSLT Source** view of a stylesheet, you can click **Find**  in the Stylus Studio tool bar. In the **Find** dialog box that appears, specify the string you want to search for and click **Find Next**. Stylus Studio highlights the first occurrence of the string you entered. In the **Text** view, you can specify that you want Stylus Studio to highlight all instances.

Depending on which view you are examining, Stylus Studio allows you to specify constraints on the search. The constraints you can specify include the following:

- Match whole word only
- Match case
- Find a regular expression
- Search inside only element tags, only element values, only attribute names, and/or only attribute values

To learn more about regular expression syntax, visit <http://www.boost.org/libs/regex/doc/syntax.html>.

Updating DOM Tree Structures

To update the DOM tree for an XML document, click the **Tree** tab at the bottom of the window that contains the document.

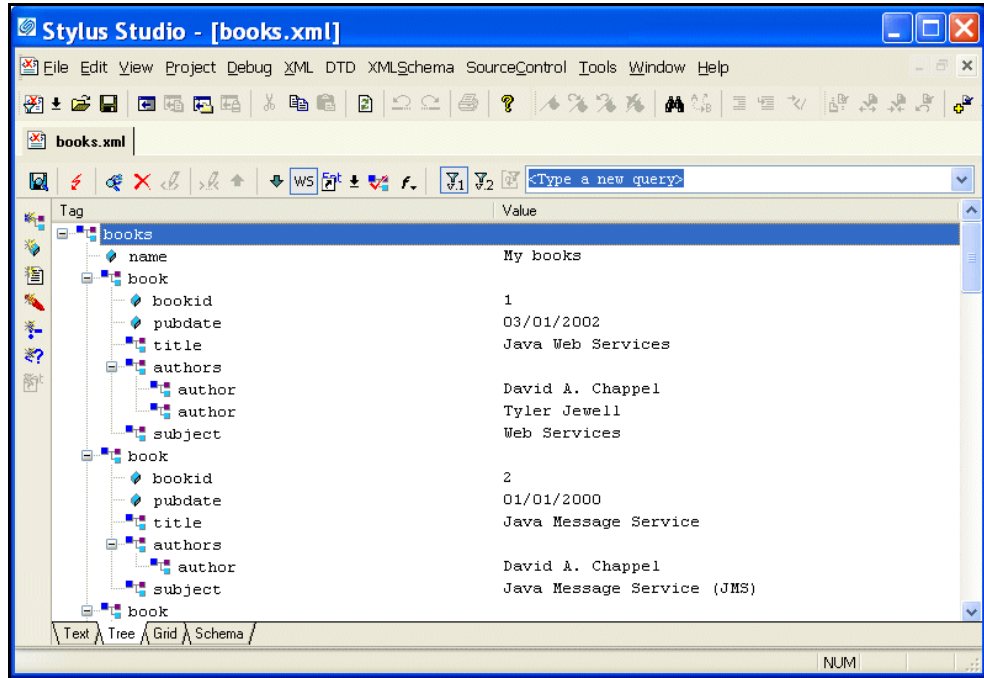



Figure 112. Tree Tab in XML Editor

While you are editing, if the display does not appear to correctly represent the current tree, click **Reload Document**  in the main tool bar. If you want to perform a certain action and Stylus Studio has grayed out the button for that action, try clicking **Refresh** first.

To save your file, select **File > Save** from the Stylus Studio menu bar or click **Save** in the Stylus Studio tool bar.


This section discusses the following topics:

- [“Displaying All Nodes in the Tree View”](#) on page 159
- [“Adding a Node in the Tree View”](#) on page 159
- [“Deleting a Node in the Tree View”](#) on page 160
- [“Moving a Node in the Tree View”](#) on page 160
- [“Changing the Name or Value of a Node in the Tree View”](#) on page 160

- “Obtaining the XPath for a Node” on page 161

Displaying All Nodes in the Tree View


To expand a tree so that you can see all the nodes in the tree, click the root node and then press the asterisk (*) key in the numeric key pad. To expand any particular node, click that node and press * in the numeric key pad.

The default tree view of your document does not include nodes that contain only blank spaces, line feeds, or tabs. To toggle between the default view and a view that does display all nodes, click **White Space**  in the Stylus Studio tool bar. This view is most helpful when you are operating on the DOM and need to know the exact structure of the tree.

Adding a Node in the Tree View

Along the left side of the window that contains your DOM tree, there are buttons that represent the types of nodes you can add to your document. The procedure for adding a node is similar for all types of nodes.

◆ To add an element:

1. Click the element that you want to be the parent of the new element, or click an element that you want to be a sibling of the new element.
2. To add a child element, click **New Element** . To add a sibling element, hold down the Shift key and click **New Element**.

Alternative: To add a child element, press Ctrl+E. To add a sibling element, press Ctrl+Shift+E.


If your XML document specifies a DTD, Stylus Studio displays a list of the elements that you can add at that location. If your document is associated with an XML Schema or does not specify a DTD, Stylus Studio prompts you to specify the name of the new element.

3. Double-click the element you want to add, or type the name of the new element and press Enter. If you added a child node, Stylus Studio adds it as the last child.
4. If the new element contains data, type a value for the new element and press Enter.

Deleting a Node in the Tree View

Along the left side of the window that contains your DOM tree, there are buttons that represent the types of nodes you can add to your document. The procedure for deleting a node is similar for all types of nodes.

◆ **To delete a node:**

1. Click the node you want to delete.
2. Click **Delete Node** .

Moving a Node in the Tree View

Along the left side of the window that contains your DOM tree, there are buttons that represent the types of nodes you can add to your document. The procedure for moving a node is similar for all types of nodes.

◆ **To move a node:**


1. Click the node you want to move.
2. Click the up and down arrows at the top of the document window to move the node up or down the tree.

Alternative: Drag the node to its new location.

Changing the Name or Value of a Node in the Tree View


Along the left side of the window that contains your DOM tree, there are buttons that represent the types of nodes you can add to your document. The procedure for renaming a node is similar for all types of nodes.

◆ **To rename a node:**

1. Click the node you want to rename.
2. Click **Change Name** . If your document specifies a DTD, Stylus Studio displays a list of the possible names. If your document does not specify a DTD, Stylus Studio opens an edit field.
3. Double-click the new name, or type the new name and press Enter.

◆ **To change the value of a node:**

To change the value of a node:

1. Click the node whose value you want to change.
2. Click **Change Value** . Stylus Studio displays an update field.
3. Type the new value and press Enter.

Obtaining the XPath for a Node

◆ **To obtain the XPath expression that returns a particular node:**

1. In the XML editor, click the **Tree** tab.
2. Right-click the node for which you want the XPath expression.
3. In the shortcut menu that appears, click **Copy XPath Query to Clipboard**.
4. Press Ctrl+V to paste the XPath expression where you want it.

Using the Grid Tab



The XML Editor **Grid** tab is available only in Stylus Studio XML Enterprise Suite and Stylus Studio XML Professional Suite.

The **Grid** view of an XML document is useful for structured data – it is a convenient way to view and work with documents that contain multiple instances of the same type of element.

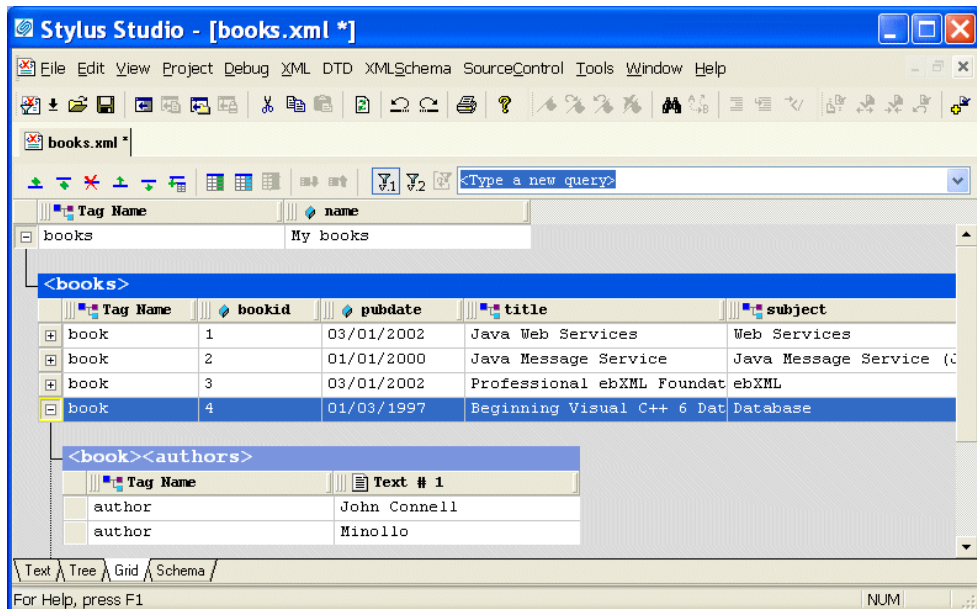


Figure 113. Grid View of books.xml

This section describes the features of the **Grid** tab and how to use it to edit XML documents. This section covers the following topics:

- “[Layout of the Grid Tab](#)” on page 163
- “[Features of the Grid Tab](#)” on page 163
- “[Moving Around the Grid Tab](#)” on page 167
- “[Working with Rows](#)” on page 169
- “[Working with Columns](#)” on page 171

- “Working with Tables” on page 173



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the XML Grid Editor video](#).

A complete list of the videos demonstrating Stylus Studio’s features is here:
http://www.stylusstudio.com/xml_videos.html.

Layout of the Grid Tab

The **Grid** tab consists of a tool bar and a display area. The tool bar has buttons to perform actions and operations on both the grid itself and on the underlying XML document represented in the grid. An example of the former is the ability to show the child elements of the document’s root element; they are hidden by default. An example of the latter is the ability to add a new instance of an element or to change a value. These operations are also accessible from the **XML > Grid Editing** menu, as well as from the grid shortcut menu (right-click on the grid).

The tool bar also includes a query field, which allows you to enter an XPath expression to query the XML document. Results are displayed in the **Query Output** window, which appears when you run the query if it is not already displayed. See “[Querying XML Documents Using XPath](#)” on page 211 for more information on this feature.

The display area shows the XML document, both its structure and content, rendered in a tabular, or grid format.

Features of the Grid Tab

This section describes the features of the **Grid** tab. It covers the following topics:

- “[Expanding and Collapsing Nodes](#)” on page 164
- “[Collapsing Empty Nodes](#)” on page 164
- “[Renaming Nodes](#)” on page 166
- “[Resizing Columns](#)” on page 166
- “[Showing Row Tag Names](#)” on page 167

Expanding and Collapsing Nodes

When you first display a document in the **Grid** tab, the document is collapsed so that it shows just the root element (here it is <books>) and its name attribute (My books), as shown in [Figure 114](#).

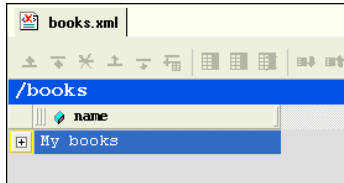


Figure 114. Default Display – Document Elements Are Collapsed

A plus sign displayed to the left of the node name indicates that this node has child nodes. You can click the plus sign to display a subgrid that displays the child nodes, as shown in [Figure 115](#).

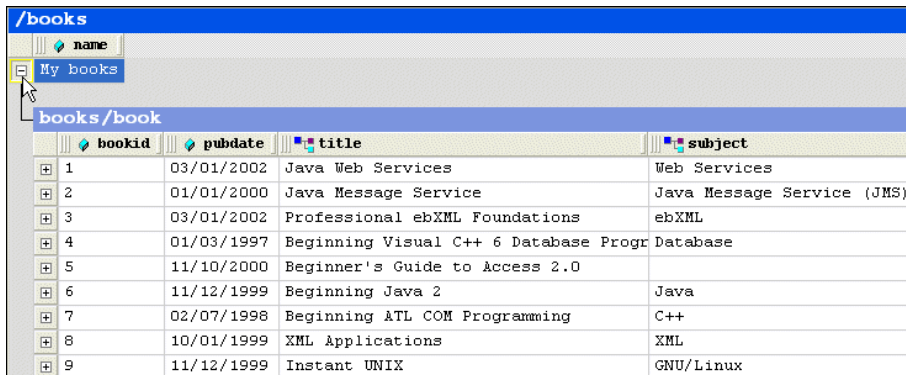


Figure 115. Click Plus Signs to Expand Collapsed Tables

You can continue to drill down in this fashion to view all values.

- ◆ To expand a node, click the plus sign (⊕).

Collapsing Empty Nodes

Some nodes in a document are simply containers – they have no content of their own. An example of a container node is the <authors> element in books.xml. The <authors>

element is simply a container for one or more <author> elements, as shown in this excerpt of books.xml:

```
<authors>
  <author>David A. Chappel</author>
  <author>Tyler Jewell</author>
</authors>
```

To streamline the display, Stylus Studio hides the tables that represent container nodes. Information about container nodes is displayed in the child node's header. [Figure 116](#) shows the default display for the author element. Notice that the header, book/authors/author, contains information about the container node, authors.

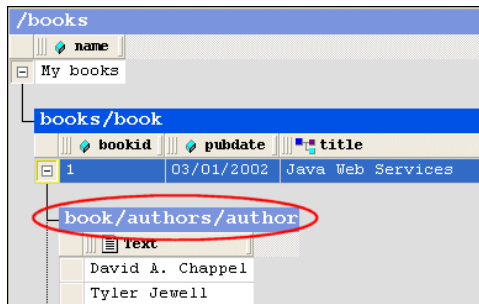


Figure 116. Table Headers Show Full Path

If you want, however, you can display the tables associated with container nodes, as shown in [Figure 117](#).

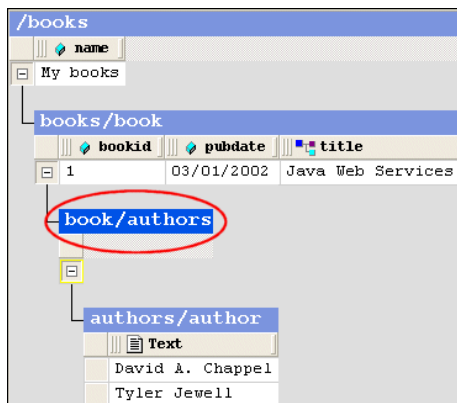


Figure 117. Container Nodes Are Hidden by Default

The table associated with the authors node now appears in the grid; it is empty (it has no rows) because it is a container. The elements it contains are displayed in their own table, authors/author.

- ◆ **To display container nodes, click Simplified View** ()

This action is also available from the **XML > Grid Editing** menu and from the grid shortcut menu.

Renaming Nodes

You can rename container nodes directly in the grid.

- ◆ **To rename a node:**
 1. Double-click the header that represents the node you want to rename.
The node name is selected.
 2. Type the name you want to use for the node.
 3. Press Enter (or click elsewhere in the grid or grid background).

Resizing Columns

When you expand a node, Stylus Studio displays it in uniform columns. You can resize columns to any width you prefer by dragging the handle on the right side of the column header, as shown in [Figure 118](#).

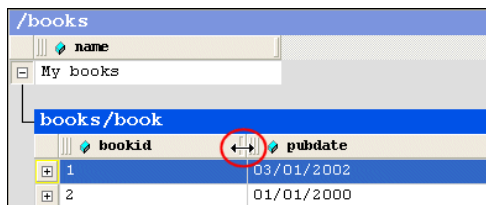


Figure 118. Resize Columns by Dragging the Right Handle

- ◆ **To resize a column, drag the handle on the right side of the column header.**

Showing Row Tag Names

In the grid view of a structured XML document, each child element of a node corresponds to a row in a table. For example, the <books> node of `books.xml` contains nine child elements; each row is an instance of the <book> element. To preserve space in the grid, the tag names of child elements are not displayed as a separate column in the table. Rather, as shown in [Figure 115](#), this information is displayed in the table header itself.

If you want, you can display the tag name for child elements in their own columns, as shown in [Figure 119](#).

Tag Name	name
books	My books

Tag Name	bookid	pubdate	title
book	1	03/01/2002	Java Web Services
book	2	01/01/2000	Java Message Service
book	3	03/01/2002	Professional ebXML Foundations
book	4	01/03/1997	Beginning Visual C++ 6 Database Progr
book	5	11/10/2000	Beginner's Guide to Access 2.0
book	6	11/12/1999	Beginning Java 2
book	7	02/07/1998	Beginning ATL COM Programming
book	8	10/01/1999	XML Applications
book	9	11/12/1999	Instant UNIX

Figure 119. Displaying the Root's Child Element

- ◆ To toggle the display of child element names, click **Toggle Row Tag Name** ()

This action is also available from the **XML > Grid Editing** menu.

Moving Around the Grid Tab

You can move around the grid using the mouse (click where you want to go) and the keyboard. Keyboard navigation is presented in the following table.

Table 14. Keyboard Navigation in the Grid

Key	Action
Up/Down arrow keys	Moves the row highlight in the direction of the arrow key you press.
Left/Right arrow keys	Moves the focus from one cell to the next, in the direction of the arrow key you press.
Page Up	Moves the row highlight to the root node's attribute.

Table 14. Keyboard Navigation in the Grid

Key	Action
Page Down	Moves the row highlight to the last row in the document.
Tab	Moves the focus forward to the next cell in the row; moves to the first cell of the next row when you hit the last cell in a row.
Shift + Tab	Moves the focus backward to the previous cell in the row; moves to the last cell of the previous row when you hit the first cell in a row.

Selecting Items in the Grid

When you select a cell in a table:

- The row is selected; you can perform row-oriented actions like changing the row's order in the table. You can also select a row by clicking the plus sign to the left side of the row.
- The column is selected; you can perform column-oriented actions like adding a new column or renaming an existing one.
- The cell gets focus.

Tip Pressing Enter places a selected cell in Edit mode.

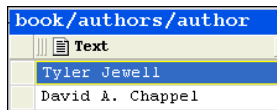
How Grid Changes Affect the XML Document

When you make a change to the document structure or content on the **Grid** tab, those changes are reflected immediately in the underlying XML document. You can see your changes affect the document on the **Text** tab.

Consider the following excerpt from `books.xml`.

```
<authors>
  <author>David A. Chappel</author>
  <author>Tyler Jewell</author>
</authors>
```

If you move the rows in the authors table, for example, as shown in [Figure 120](#),



book/authors/author	
	Text
	Tyler Jewell
	David A. Chappel

Figure 120. Moving a Row Affects XML

the underlying XML changes accordingly:

```
<authors>
  <author>Tyler Jewell</author>
  <author>David A. Chappel</author>
</authors>
```

Types of Changes that Affect the Document

The following changes, all of which can be made using **Grid** tab, affect the underlying XML document:

- Adding, deleting, reordering rows
- Adding, deleting, reordering, and renaming columns
- Adding, deleting, reordering, and sorting tables
- Changing element and attribute values
- Renaming container elements

Changes you make affect the current instance only. For example, in the example shown in [Figure 120](#), only that instance of the nested table is affected. If you add a column to books/book, however, every instance of books/book gets that new column.

Working with Rows

Stylus Studio provides several features to help you work with table rows in the **Grid** tab. Changes you make to tables in the **Grid** tab, such as adding a new row or modifying a value, are reflected in the underlying XML document.



This section covers the following topics:

- [“Reordering Rows”](#) on page 170
- [“Adding and Deleting Rows”](#) on page 170

Reordering Rows

You can move rows up and down within the same table. Changes you make to row order affect the element order in the underlying XML document.

◆ **To move a row:**



1. Select the row you want to move.
2. Click the **Move Up** () or **Move Down** () button to move the row to the desired location in the table. These actions are also available from the **XML > Grid Editing** menu and from the grid shortcut menu.

Adding and Deleting Rows

You can add and delete rows in a table. Changes you make to the table in this way affect the number of instances of the element in the table. When you add a row, you can insert it in the table above or below the currently selected row.


Tip You can move rows up and down within a table.

◆ **To add a row:**

1. Select the row next to which you want to insert a new row.
2. Click the **Insert Row Before** () or **Insert Row After** () button to add the new row to the table. These actions are also available from the **XML > Grid Editing** menu and from the grid shortcut menu.

The row is added to the table.

◆ **To delete a row:**

1. Select the row you want to delete.
2. Click **Delete** (). This action is also available from the **XML > Grid Editing** menu and from the grid shortcut menu.

The row is deleted from the table.

Working with Columns

Stylus Studio provides several features to help you work with table columns in the **Grid** tab. Changes you make to tables in the **Grid** tab, such as adding a new column or reordering existing columns, are reflected in the underlying XML document.

This section covers the following topics:

- “[Selecting a Column](#)” on page 171
- “[Adding Columns](#)” on page 171
- “[Deleting Columns](#)” on page 172
- “[Reordering Columns](#)” on page 172
- “[Renaming Columns](#)” on page 173
- “[Changing a Value](#)” on page 173

Selecting a Column

Column operations can be performed when you select any cell in a column. When a cell (and, therefore, its column) is selected, it is highlighted with a yellow outline. As shown in [Figure 121](#), the `<title>` column is selected – the cell containing Java Message Service is the one that is highlighted.

books/book				
	bookid	pubdate	title	subject
1		03/01/2002	Java Web Services	Web Services
2		01/01/2000	Java Message Service	Java Message Service (JMS)
3		03/01/2002	Professional ebXML Found	ebXML



Figure 121. Selected Cells are Highlighted in Yellow

- ◆ **To select a column, click any cell in the column you wish to select.**

Adding Columns


You can add two types of columns to tables in the **Grid** tab – attribute columns and element columns. The procedure for adding both types of columns is the same. When you add a column, it is inserted immediately after the last column of its type. You can move columns after you create them.

◆ To add a column:

1. Select the row in which you want to add a column.
2. Click **Add Attribute Column** () or **Add Element Column** (). These actions are also available from the **XML > Grid Editing** menu and from the grid shortcut menu. The column is added to the table.
3. If you want, move the column to a new location in the row. See [“Reordering Columns”](#) on page 172.

Deleting Columns

◆ To delete a column:

1. Select a cell in the column you want to delete.
A yellow border appears around the cell you select.
2. Click **Delete Column** (). This action is also available from the **XML > Grid Editing** menu and from the grid shortcut menu. The column is deleted from the table.

Reordering Columns

You can reorder columns in the grid by dragging them to the position you desire.

◆ To reorder a column:

1. Place the pointer on the left handle in the column header.
2. Press and hold mouse button one.
The cursor changes shape, as shown here.

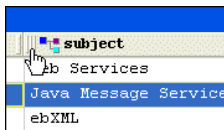


Figure 122. Moving a Column

3. Drag the column to the location in the row you want.
4. Release the mouse button.
The column is placed in the new location within the row.

Renaming Columns

You can rename columns in the grid. This has the effect of renaming the corresponding attribute or element name in the underlying XML document.

Note You cannot rename the root element from the **Grid** tab.

◆ **To rename a column:**

1. Select a cell in the column you want to rename.
A yellow border appears around the cell you select.
2. Click **Rename Column** (📄). This action is also available from the **XML > Grid Editing** menu and from the column shortcut menu.
The column is renamed.

Changing a Value

You can change element and attribute values.

◆ **To change a value:**

1. Double-click the cell whose value you want to change.
The cell field becomes editable, as shown here.

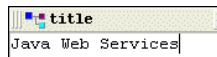


Figure 123. Changing a Value

2. Edit the value as required.
3. Press Enter.

Working with Tables

Stylus Studio provides several features to help you work with tables in the **Grid** tab. Changes you make to tables in the **Grid** tab, such as adding a nested table, are reflected in the underlying XML document.

This section covers the following topics:

- [“Adding a Nested Table”](#) on page 174
- [“Moving a Nested Table”](#) on page 175
- [“Deleting a Table”](#) on page 175

- “[Sorting a Table](#)” on page 176
- “[Copying a Table as Tab-Delimited Text](#)” on page 176

Adding a Nested Table

You add nested tables to a document in the **Grid** tab using the **Add Nested Table** dialog box, shown in [Figure 124](#). This dialog box allows you to specify the path to the root for the new table, a row element name, and the number of rows.

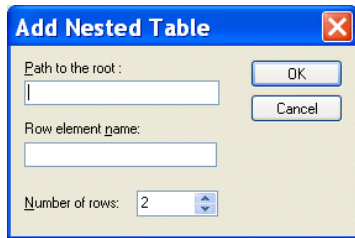


Figure 124. Add Table Dialog Box

A nested table is created as a child of the current element. The nested table shown in [Figure 125](#), myTable, was created as a child of the <book> element.




Figure 125. Default Nested Table

Nested tables are created with two default rows, which use the element name you provide in the **Row Element Name** field of the **Add Nested Table** dialog box. Rows get a default

text value of Row n text, where n is an incrementing value starting with 1. You specify the number of rows using the **Number of rows** field.

◆ **To add a nested table:**



1. Select the element to which you want to add a nested table.
2. Click **Add Nested Table** () . This action is also available from the **XML > Grid Editing** menu and from the grid shortcut menu.
The **Add Nested Table** dialog box appears.
3. Optionally, specify the path to the root. If you leave this field blank, the nested table is created as a child of the current element.
4. Enter a row element name.
5. Optionally, change the number of default rows.
6. Click OK.

The nested table is added to the document and appears in the grid.

Moving a Nested Table


You can change the order of nested tables within a row.

◆ **To move a nested table:**

1. Select the heading of the nested table you want to move.
2. Click the **Move Up** () or **Move Down** () button to move the table to the desired location. These actions are also available from the **XML > Grid Editing** menu and from the grid shortcut menu.

Deleting a Table

◆ **To delete a table:**



1. Select the heading of the table you want to delete.
2. Click **Delete** () . This action is also available from the **XML > Grid Editing** menu and from the grid shortcut menu.

The table is deleted from the document.

Sorting a Table

You can sort tables on any column in ascending or descending order.

◆ **To sort a table:**

1. Select a cell in the column on which you want to sort the table.
2. Click the **Sort Ascending** () or **Sort Descending** () button to sort the table rows in ascending or descending order, respectively. These actions are also available from the **XML > Grid Editing** menu and from the grid shortcut menu.

Tip

You can also display sort options by right-clicking the column heading.

The table rows are sorted based on the order you select.

Copying a Table as Tab-Delimited Text

You can copy a tab-delimited text version of a table to the clipboard. This makes it possible to paste document contents from the grid into spreadsheets and other editors that can manage tab-delimited files. [Figure 126](#) shows books/book in books.xml pasted into Microsoft Excel, for example.

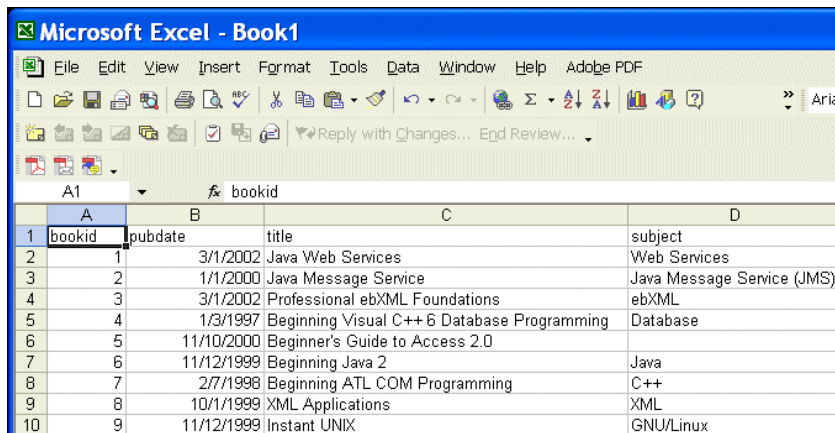


Figure 126. Pasting a Table into a Spreadsheet

Note that when you use this feature, the entire table is copied – column headings (element and attribute names) are not distinguished from cell contents (element and attribute values) in the spreadsheet.

◆ **To copy a tab-delimited table to the clipboard:**

1. Select the heading of the table you want to copy.
2. Select **XML > Grid Editing > Copy as Tab-Delimited** from the menu. This action is also available from the grid shortcut menu.

Diffing Folders and XML Documents



XML differencing is available only in Stylus Studio XML Enterprise Suite and Stylus Studio XML Professional Suite.

During application development, it can be useful to be able to compare two or more XML documents, or to compare the contents of two folders, in order to identify the type and number of differences between them. The process of comparing one document (or folder) with another is referred to as *diffing*. Stylus Studio provides utilities for diffing folders and documents.



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the XML Diff video](#).

A complete list of the videos demonstrating Stylus Studio's features is here: http://www.stylusstudio.com/xml_videos.html.

This section covers the following topics:

- “[Overview](#)” on page 178
- “[Diffing Folders](#)” on page 183
- “[The XML Diff Viewer](#)” on page 187
- “[Diffing a Pair of XML Documents](#)” on page 194
- “[Diffing Multiple Documents](#)” on page 196
- “[Modifying Default Diff Settings](#)” on page 201
- “[Running the Diff Tool from the Command Line](#)” on page 205

Overview

Stylus Studio’s Diff tool lets you easily compare two or more versions of the same document in the XML Diff Viewer (as shown in [Figure 127](#)), or the contents of two folders (as shown in [Figure 131](#)).

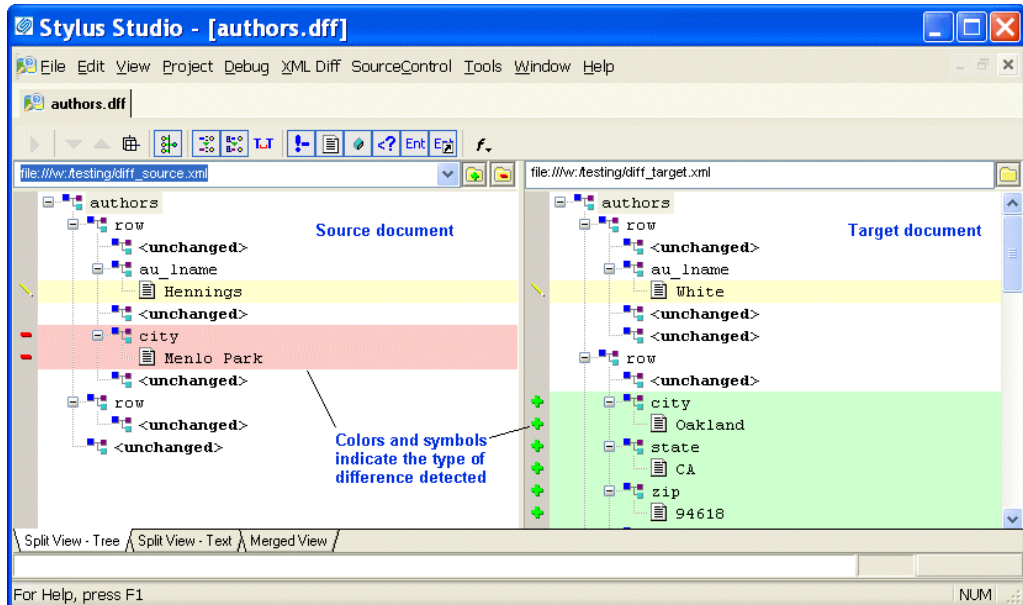


Figure 127. Example of Dified Documents in the XML Diff Viewer

Customizable color-coding lets you quickly determine how one document differs from another – green, for example, identifies objects (such as elements and attributes) that are present in the target document, but which do not exist in the source document. When you hover the pointer over symbols displayed in the side bars of the source and target document windows, Stylus Studio displays a tool tip that indicates the specific nature of the change.

This section covers the following topics:

- “Sources and Targets” on page 179
- “The Diff Configuration File” on page 179
- “What Diffs Are Calculated?” on page 179
- “Tuning the Diffing Algorithm” on page 180
- “When Does the Diff Run?” on page 181

- [“Running the Diff Manually”](#) on page 182
- [“Symbols and Background Colors”](#) on page 182

Sources and Targets

When you use Stylus Studio to diff documents (or folders), you select a *source* and a *target*. Stylus Studio considers the source document or folder to be the baseline, or current standard; the target document or folder is assumed to be some other version (it might be older or newer, for example) of the source. The Stylus Studio Diff tool illustrates how this other version, the target, differs from the source (or sources) you have selected.

Tip You can open source and target documents in the XML Editor from the XML Diff Viewer – right click on the XML Diff Viewer background and select the document you want to edit from the short-cut menu of source and target documents displayed by Stylus Studio. This feature is context-sensitive – if you right-click on a node that has been removed, the target document will not be listed, for example.

The Diff Configuration File

You can save the information associated with a given XML diff session in a *diff configuration file*. Diff configuration files make it easy to perform a diff on the same set of XML documents over time. Examples of the information saved with the diff configuration file include the URLs of the source and target documents, and any settings made on the **XML Diff** menu or tool bar. Diff configuration files are created with a `.dff` extension.

Changes made to the source and target documents are detected by Stylus Studio the next time you open the diff configuration file, allowing you to diff the files at that time. (Whether or not the diff is run automatically when you open the diff configuration file depends on **Autorun Diff** settings on the **Engine** page of the **Options** dialog box. See [“When Does the Diff Run?”](#) on page 181 for more information.)

What Diffs Are Calculated?

This section describes how Stylus Studio diffs XML documents and folders.

Documents – The Stylus Studio Diff engine compares source and target documents in their entirety. If you want, you can use the **Engine** page of the **Options** dialog box to exclude certain items from the diff calculation. These items include:

- Comments
- Text

- Entities
- Attributes
- Processing instructions

You can also specify whether or not you want Stylus Studio to:

- Use URIs to compare namespaces
- Expand entity references
- Ignore text formatting characters (new lines, carriage returns, and tabs)

See [“Modifying Default Diff Settings”](#) on page 201 to learn how to set these and other diff options.

Folders – Options for diffing XML documents do not affect how Stylus Studio diffs folders. When diffing folders, Stylus Studio compares one folder’s contents with another. See [“Diffing Folders”](#) on page 183 for more information on this topic.

Tuning the Diffing Algorithm

The purpose of any diffing tool is to identify the list of logical operations required to change the source document into the target document. Examples of logical operations include additions, revisions, and deletions. Even diffs between simple XML documents can yield a long list, sometimes with redundant operations. Ideally, the list of operations should be reduced to make it as economical as possible; that is, the list should be able to answer the question, *What are the fewest number of changes required to turn the source into the target?*

Calculating such a list can be time-consuming and resource intensive, and these costs might not be worth the benefits to a given user. For this reason, Stylus Studio provides settings that let you tune the diffing algorithm used by the XML Diff engine. Tuning settings are displayed in the **Performance** group box on the **Engine** page of the **Options** dialog box.

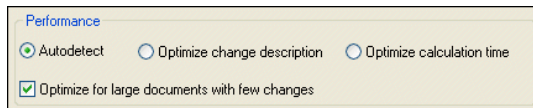


Figure 128. Performance Settings Let You Tune the Diffing Algorithm

You can

- Select a tuning that optimizes the algorithm to provide the most economical set of changes possible (**Optimize change description**). As described earlier, this calculation, though it yields the best results, can be costly in terms of time and processing resources.
- Select a tuning that optimizes the algorithm to provide the set of changes in the shortest time possible (**Optimize calculation time**).
- Let Stylus Studio decide (**Autodetect**). By default, Stylus Studio tries to provide the most economical set of changes possible, but if it determines that processing resources are limited or that the calculation will take too much time, it reverts to the algorithm tuning that is optimized for speed.

Handling Large Documents

The **Optimize for large documents with few changes** setting helps speed the diffing of large (greater than 1MB) documents. This setting can be used in conjunction with any of the algorithm tuning settings and is on by default.

When Does the Diff Run?

Stylus Studio runs the diff automatically, as soon as you specify the target document or folder. Whether or not subsequent changes cause Stylus Studio to automatically recalculate the diff is determined by the **Autorun Diff** settings on the **Engine** page of the **Options** dialog box. Changes that can make a diff recalculation necessary include adding new source and target documents, changing the underlying source and target documents themselves, or to changes to certain **Engine** settings.

Options That Affect When the Diff Runs

These settings, on the **Engine** page of the **Options** dialog box, determine when and whether Stylus Studio automatically recalculates the diff.

- **On changes** – Certain types of changes to the diff configuration file require Stylus Studio to recalculate the diff. These changes include:
 - Adding a new source document
 - Changing the target document
 - Changing the **Use URI to compare namespaces** setting
 - Changing the **Expand entity references** setting


If the **On changes** setting is on, Stylus Studio automatically runs the diff when any of these changes occurs.

- **If files modified** – If you make and save changes to a source or target document, Stylus Studio automatically runs the diff if this setting is on.

Note These settings do not affect the diffing of folders.

See [“Modifying Default Diff Settings”](#) on page 201 to learn more about setting these and other Diff options.

Running the Diff Manually





You can run the diff manually by clicking the Calculate diff button (). Stylus Studio activates this button when it detects the need to recalculate the diff, and the **On changes** or **If files modified** settings are off. These settings, as described in [“When Does the Diff Run?”](#) on page 181, cause Stylus Studio to run the diff automatically.

You can also run the diff from the command line. See [“Running the Diff Tool from the Command Line”](#) on page 205.

Symbols and Background Colors

Stylus Studio uses symbols and background colors to alert you to differences in diffed documents and folders. The following table summarizes the symbols and default background colors, and the types of changes they represent.

Table 15. Default Colors Used for Diffing Files and Folders

<i>Symbol</i>	<i>Background Color</i>	<i>Identifies</i>
	Light green	Added items; appears in the target document and identifies an item that is present in the target but absent from the source
	Light red	Removed items; appears in the source document and identifies an item that is present in the source but absent from the target.
	Light yellow	Changed items; can appear in both source and target documents.
	Light gray	Collapsed item containing changes (such as an added, changed, or removed node).

You can change the background colors on the **Presentation** page of the **Options** dialog box.

Combined Symbols

As described in [Table 15, Default Colors Used for Diffing Files and Folders](#), Stylus Studio displays a turquoise block (■) when a node that you have collapsed contains changes. Sometimes, the node itself has changes. In this case, Stylus Studio combines two symbols – one indicating the change of a child within the collapsed node, and one to the node itself. Consider the following illustration:



Figure 129. Sample of a Collapsed Node with Changes

Here, the `city` node displays a combined symbol – the turquoise box indicates that a change exists within the collapsed node; the minus sign indicates that the `city` node is not present in the source document. Expanding the `city` node makes the scope and nature of the changes explicit:



Figure 130. Expanded Node with Changes

Tip Hover the mouse point over these symbols to display tool tips that describe the nature of the change.

Additional Symbols for Diffing Multiple Sources

Stylus Studio uses other symbols in the target document window when you diff multiple source documents. See [“Symbols Used in the Target Document Window”](#) on page 197.

Diffing Folders

Stylus Studio allows you to diff two folders. As shown in [Figure 131](#), the **Diff Folders** dialog box displays the contents of each folder; symbols and colors, described in

“Symbols and Background Colors” on page 182, identify the types of changes in the respective folders.

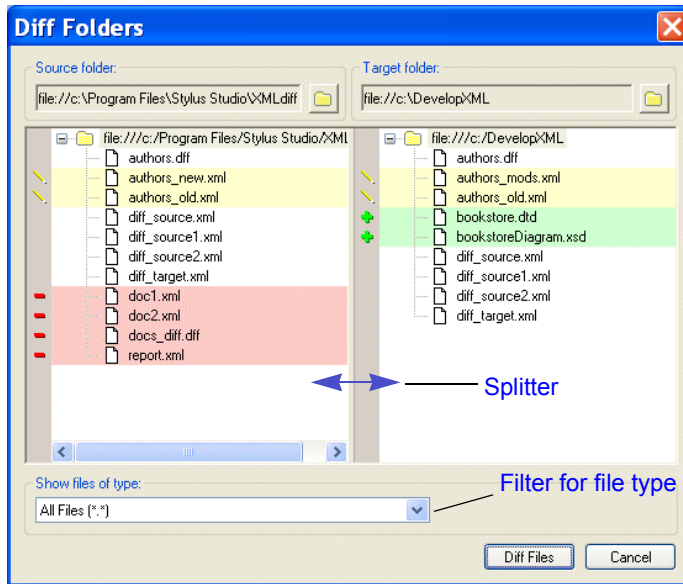


Figure 131. The Diff Folders Dialog Box After a Diff

This section covers the following topics:

- “Features” on page 184
- “How to Diff Folders” on page 185
- “How to Diff Documents from the Diff Folders Dialog Box” on page 187

Features

The **Diff Folders** dialog box has several features that make it easy to diff folders and the XML documents they contain:

- A *splitter* lets you change the width of the source and target folder windows. This can be especially useful if you are working with a folder that has nested directories.
- A *file type filter* limits the display to files with a .xml extension; if you choose, you can display (and diff) all file types, as shown in [Figure 131](#).
- An **Abort** button, shown here, appears at the bottom of the **Diff Folders** dialog box if the operation you are performing (loading or diffing a directory with a large number

of files, for example) is taking more time than usual. Clicking the **Abort** button cancels the operation.

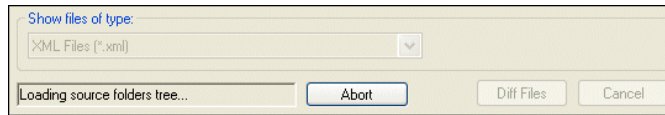


Figure 132. Abort Button Lets You Cancel Long Load Operations

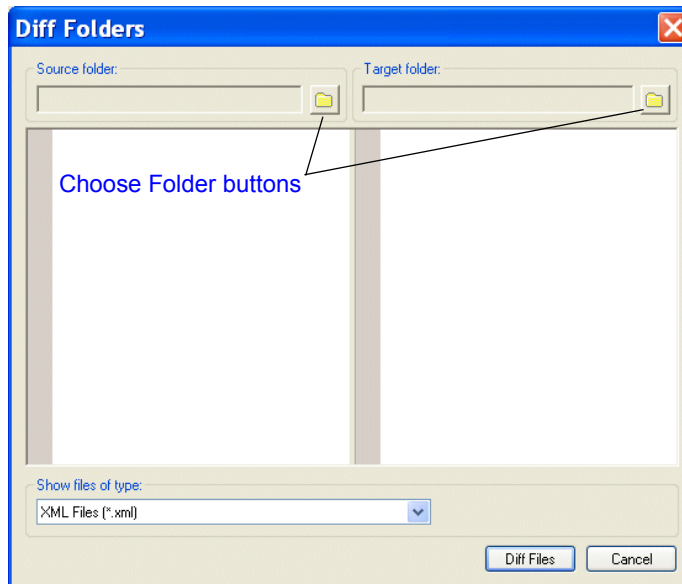
Tip This feature is also part of the XML Diff Viewer.

- The **Diff Files** button allows you to perform a diff of XML documents in the source and target folders. See [“How to Diff Documents from the Diff Folders Dialog Box”](#) on page 187 for more information on this topic.

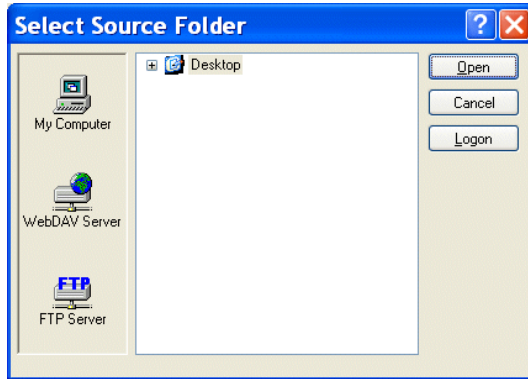
How to Diff Folders

◆ **To diff folders:**

1. Select **Tools > Show Differences In > Folders** from the Stylus Studio menu. The **Diff Folders** dialog box appears.



2. Click the **Choose Source Folder** button ().
The **Select Source Folder** dialog box appears.



3. Expand the **Desktop** tree and navigate to the folder you want to use as the source folder for the diff.
4. Click **Open**.
The folder is displayed in the **Source folder** window of the **Diff Folders** dialog box.
5. Repeat [Step 2](#) through [Step 4](#) for the target folder.
Stylus Studio performs the diff as soon as you select the target folder for comparison.
6. Optionally, use the **Show files of type** drop-down list to filter the display to show only those files of the type you specify. (By default, Stylus Studio shows XML files – files with a .xml extension.)

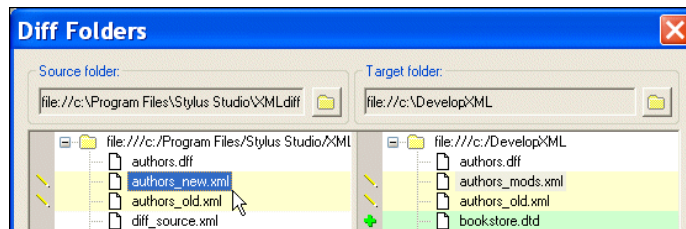
How to Diff Documents from the Diff Folders Dialog Box

You can diff XML documents in the source and target folders directly from the **Diff Folders** dialog box.

◆ **To diff two files from the Diff Folders dialog box:**

1. Click the file you want to diff.

The document is shown as selected in both the **Source folder** and **Target folder** windows. In this illustration, the document `authors_new.xml` was selected.



Tip Notice that, even though the file names are different, Stylus Studio is able to infer that `authors_new.xml` and `authors_mods.xml` are actually the same document.

If you select a document that cannot be diffed, you will not see the selection in the opposite window.

2. Click the **Diff Files** button.

Stylus Studio displays the XML Diff Viewer window.

For more information on diffing documents, see [“Diffing a Pair of XML Documents”](#) on page 194.

The XML Diff Viewer

This section describes the XML Diff Viewer and its features, including the different views available for comparing documents, the XML Diff Viewer tool bar, and tools for loading documents.

This section covers the following topics:

- [“Split View - Tree”](#) on page 188
- [“Split View - Text”](#) on page 189
- [“Merged View”](#) on page 190

- “View Symbols and Colors” on page 191
- “The XML Diff Viewer Tool Bar” on page 191
- “Tools for Working with Documents” on page 194

Split View - Tree

You use the XML Diff Viewer to compare two or more XML documents. By default, the XML Diff Viewer displays the diffed documents on the **Split View - Tree** tab. This view, shown in Figure 133, shows the documents side-by-side using a tree/node representation.

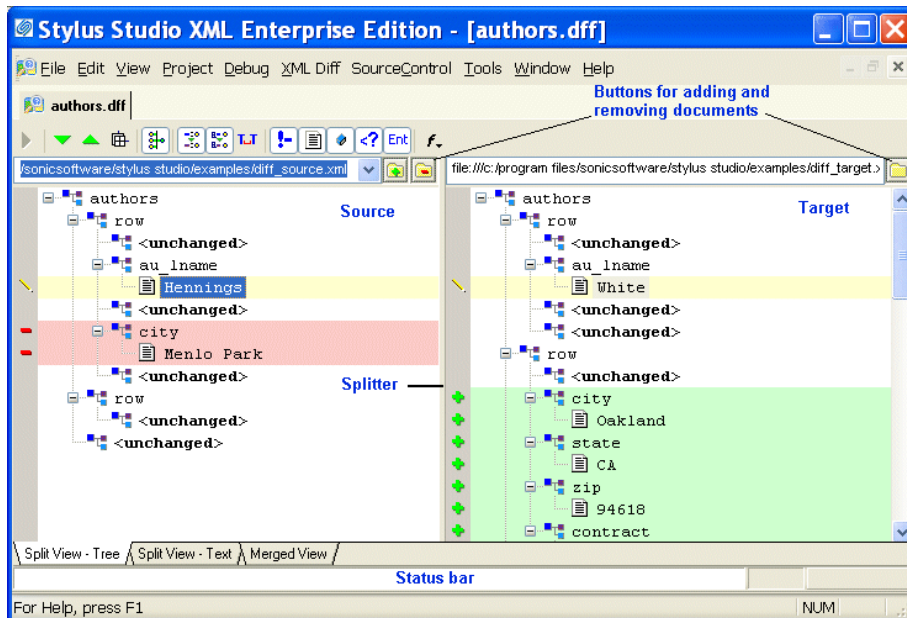


Figure 133. XML Diff Viewer – Split View - Tree

In split views (there is also a split view that shows documents in XML), source documents are displayed on the left, the target document on the right. A *splitter* between the two panes allows you to change the width of the source and target document panes by dragging the splitter to the left and right.

Split View - Text

The **Split View - Text** tab also shows source and target documents side-by-side in plain XML.

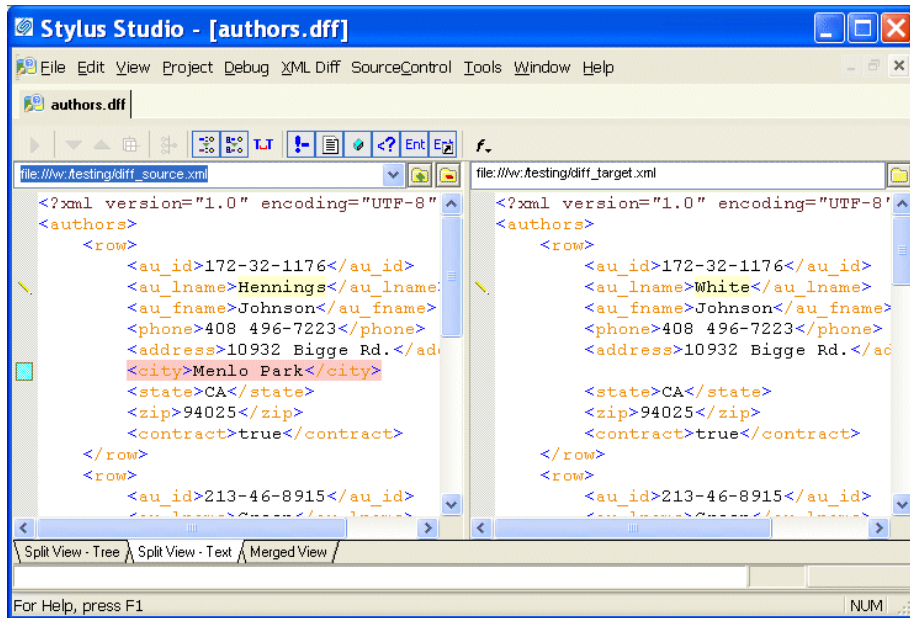


Figure 134. XML Diff Viewer – Split View - Text

Merged View

If you prefer, you can select the **Merged View** tab, which folds the nodes from the source and target documents into a single window, as shown in [Figure 135](#).



Figure 135. XML Diff Viewer – Merged View

The merged view displays changed items in pairs – the item from the target document appears first, the item from the source document is shown second, as shown in [Figure 136](#).

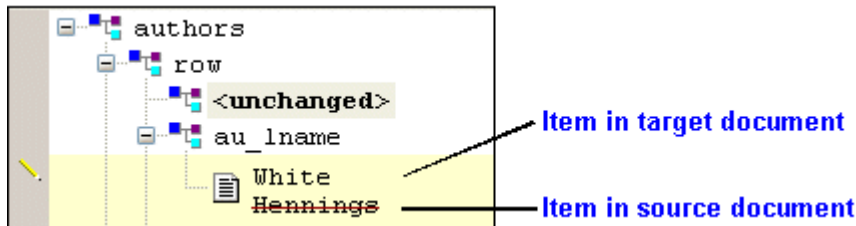


Figure 136. Close-up of Merged View

In this example, the line through the <au_lname> element in the source document, Hennings, indicates that it has changed to White in the target document.

View Symbols and Colors

All views use the same symbols and color schemes to identify the types of changes detected by the Stylus Studio diff calculation – by default, green for added items, yellow for changed items, and red for removed items. In addition, the text font and size are controlled by the settings for the XML Editor on the **Editor Format** page of the **Options** dialog box.

See “[Symbols and Background Colors](#)” on page 182 for more information on this topic, and to learn how you can assign custom colors to the results of standard differencing operations.

The XML Diff Viewer Tool Bar

The XML Diff Viewer tool bar, shown in [Figure 137](#), provides tools to help you

- Manually start the diff calculation
- Navigate source and target documents
- Change default display and diff settings
- Show or ignore differences in document items such as text nodes and attributes



Figure 137. The XML Diff Tool Bar

The following table identifies the individual tools and tells you where to find more information.

Table 16. XML Diff Tool Bar Buttons



<i>Tool Button</i>	<i>Description</i>
	Calculates the differences in the documents you have selected. This button is active only when Stylus Studio detects the need to calculate differences. This button is disabled if you have selected On changes and If files modified settings. See “ Engine Settings ” on page 203.
	Skips to the next (previous) diff in the currently selected document. You must select a line in the document to enable these buttons.

Table 16. XML Diff Tool Bar Buttons














Tool Button	Description
	By default, Stylus Studio displays collapsed documents when the diff is run. You can override this setting using the tool bar button, or you can change it permanently on the Options . page. See “ Engine Settings ” on page 203.
	By default, Stylus Studio collapses any unchanged blocks to simplify the display. You can override this setting using the tool bar button, or you can change it permanently on the Options . page. See “ Engine Settings ” on page 203.
	By default, Stylus Studio uses URIs to compare namespaces when diffing documents. You can override this setting using the tool bar button, or you can change it permanently on the Options . page. See “ Engine Settings ” on page 203. Note that changing this setting requires documents to be diffed again.
	By default, Stylus Studio expands entity references when diffing documents. You can override this setting using the tool bar button, or you can change it permanently on the Options . page. See “ Engine Settings ” on page 203. Note that changing this setting requires documents to be diffed again.
	By default, Stylus Studio considers text formatting characters (new lines, carriage returns, tabs) when diffing documents. You can override this setting using the tool bar button, or you can change it permanently on the Options . page. See “ Engine Settings ” on page 203.
	By default Stylus Studio shows differences in comments. You can override this setting using the tool bar button, or you can change it permanently on the Options . page. See “ Engine Settings ” on page 203. Note that this feature affects only the display, and not the calculation, of comment differences.
	By default Stylus Studio shows differences in text blocks. You can override this setting using the tool bar button, or you can change it permanently on the Options . page. See “ Engine Settings ” on page 203. Note that this feature affects only the display, and not the calculation, of text block differences.


Table 16. XML Diff Tool Bar Buttons

Tool Button	Description
	<p>By default Stylus Studio shows differences in attributes. You can override this setting using the tool bar button, or you can change it permanently on the Options page. See “Engine Settings” on page 203.</p> <p>Note that this feature affects only the display, and not the calculation, of attribute differences.</p>
	<p>By default Stylus Studio shows differences in processing instructions. You can override this setting using the tool bar button, or you can change it permanently on the Options page. See “Engine Settings” on page 203.</p> <p>Note that this feature affects only the display, and not the calculation, of processing instruction differences.</p>
	<p>By default Stylus Studio shows differences in entities. You can override this setting using the tool bar button, or you can change it permanently on the Options page. See “Engine Settings” on page 203.</p> <p>Note that this feature affects only the display, and not the calculation, of entity differences.</p>
	<p>By default Stylus Studio shows differences in entity references. You can override this setting using the tool bar button, or you can change it permanently on the Options page. See “Engine Settings” on page 203.</p> <p>Note that this feature affects only the display, and not the calculation, of entity differences.</p>
	<p>Allows you to change the font of documents displayed in the XML Diff Viewer.</p>

Tools for Working with Documents

The XML Diff Viewer provides several tools for working with source and target documents:

- Add/Remove document buttons. When you click the add or set document button, Stylus Studio displays the **Open** dialog box. The add button for source documents displays a green plus sign on it () to alert you to the fact that you can add multiple source documents when diffing XML documents. You can specify only a single target document, however.


You use the remove button, the folder with the red minus sign on it () , to remove the current source document from the XML diff calculation.

Tip

If Stylus Studio determines that the load or diff operation for a given XML document will take more than a moment, it displays a message and an **Abort** button at the bottom of the XML Diff Viewer window. You can click the **Abort** button to terminate the operation at any time. The message and the button are removed from the XML Diff Viewer window once the operation is complete or cancelled.

- Drop-down list. You use the drop-down list to change the current document in the XML Diff Viewer. When you change the current doc in a multi-document diff, the target document display – specifically, the symbols and colors used to identify documents – typically changes, as well. See [“Symbols Used in the Target Document Window”](#) on page 197 for more information.

Removing a Target Document

You cannot remove a target document. You can specify a *different* target document by clicking the set target button () again. This replaces the current target document with the document you select.


Diffing a Pair of XML Documents

This section describes how to use Stylus Studio to diff a pair of XML documents.

Before continuing with this section, you should read [“Overview”](#) on page 178, which describes basic information about the Stylus Studio Diff tool, and [“The XML Diff Viewer”](#) on page 187, which describes features of the XML Diff Viewer and how to use them.



How to Diff a Pair of Documents

◆ To diff a pair of documents:

1. Select **Tools > Options > Show Differences > Files** from the Stylus Studio menu. Stylus Studio displays the XML Diff Viewer.
2. In the source document window, click the add button () to add the source document. Stylus Studio displays the **Open** dialog box.

Tip

You can drag and drop a file into the entry field to load the document in the XML Diff Viewer.

3. Navigate to the document you want to load in the XML Diff Viewer.
4. Click **Open**.
5. Repeat [Step 2](#) through [Step 4](#) for the target document, using the set button for the target document window ().
By default, Stylus Studio runs the diff calculation automatically when you select the target document. If the default **On changes** setting has changed, you need to run the diff calculation manually by clicking the **Calculate diff** button (.

Diffing Multiple Documents

This section describes how to use Stylus Studio to diff multiple XML documents.

Before continuing with this section, you should read [“Overview”](#) on page 178, which describes basic information about the Stylus Studio Diff tool, and [“The XML Diff Viewer”](#) on page 187, which describes features of the XML Diff Viewer and how to use them.

This section covers the following topics:

- [“Document Focus”](#) on page 196
- [“Symbols Used in the Target Document Window”](#) on page 197

Document Focus

Diffing multiple XML documents is much the same as diffing a pair of documents – you specify the source documents (one at a time), a target document, and Stylus Studio calculates the diff.

Note When you diff *multiple* source documents against the target, Stylus Studio considers the target document to be the baseline, and the XML Diff Viewer shows how the source documents vary from the target.

When you diff multiple documents, however, only one source document can have focus at a time. Consider the following illustration, which shows three source documents (source1.xml, source2.xml, and source3.xml) and a target document (target.xml).

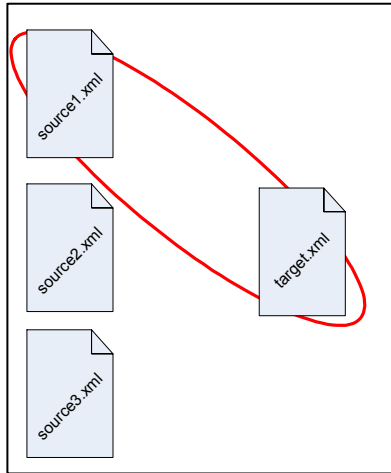



Figure 138. Example of Document Focus

In this example, the source1.xml document currently has focus. You set the focus on a given source document by selecting that document from the drop-down list at the top of the source document window.

When a source document has focus:

- Diffs are displayed for that document only, even if you have selected the **Merged View** tab for display.
- Clicking the remove document button () removes that document from the XML diff calculation.

Symbols Used in the Target Document Window

When diffing multiple documents, Stylus Studio uses an additional set of symbols in the target document window. These symbols, which are displayed in the side bar of the XML Diff Viewer window alongside the standard set of symbols described in “[Symbols and Background Colors](#)” on page 182, indicate the ways in which the change identified in the

current source document differs from changes to the same node in other source documents.

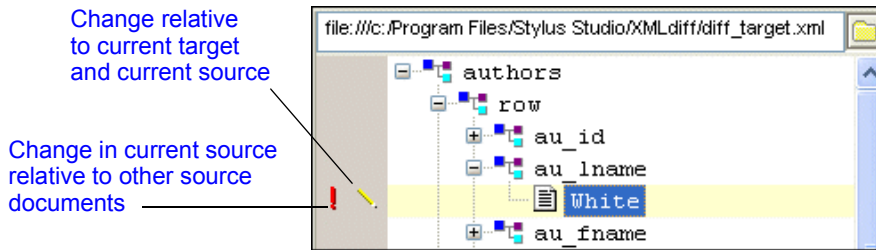




Figure 139. Example of Symbols Used When Diffing Multiple Documents

As shown in [Figure 139](#), symbols in the column closest to the document tree identify the changes relative to the currently selected source. Here, the edit symbol () indicates that the value in the target document, `White`, differs from that in the currently selected source (which happens to be `Black` in this example).

The first column of symbols characterize changes in the currently selected source relative to other source documents. Here, for example, the red exclamation point () indicates that there are conflicting modifications in other source documents – that is, other source documents contain a value other than `Black`. As shown in [Figure 140](#), when you click on a symbol in the first column, Stylus Studio displays

- A message describing the precise nature of the change
- A menu that identifies the documents in which the change occurs.

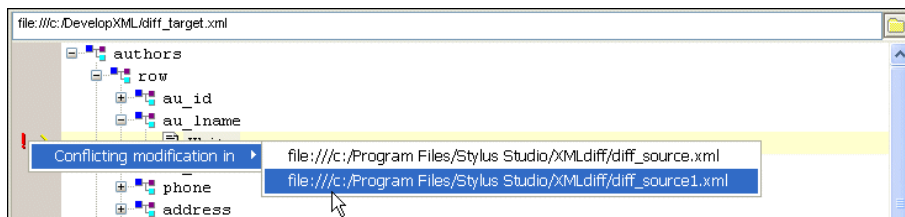






Figure 140. Easily Navigate to Changed Nodes in Other Source Documents

Clicking on a document in this menu changes the current focus to that source document, allowing you to easily navigate to the same node in a different document.

The additional symbols used by Stylus Studio when diffing multiple documents are described in [Table 17](#).

Table 17. Symbols Used to Specify Changes in Source Documents

Symbol	Description	Meaning
	Yellow circle	Modified in other documents – The object in the target and the current source document are the same, but another source document is different.
	Red exclamation point	Conflicting modification in other documents – The object in the current source differs from that in the target document. Other source documents differ from both the current source and target.
	Yellow diamond	Same modification in other documents – The source documents all differ from the target document in the same way.
	Yellow equal sign	Unchanged in other documents – The current source document differs from the target, but other source documents are the same as the target.

Consider the example in [Figure 141](#), which illustrates diffing three documents. In this example, the node in question is circled in red.

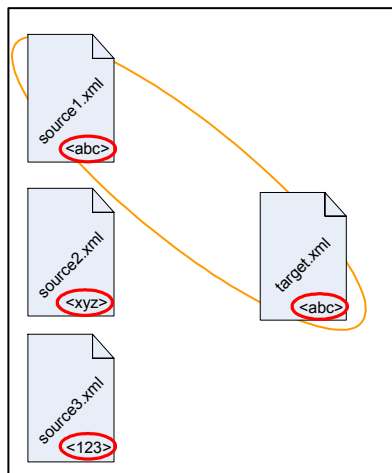






Figure 141. Diffing Three Documents

Notice that:

- source1.xml currently has focus.
- source1.xml and the target document node have the same value (<abc>)
- The node in question varies in both of the remaining source documents (it is <abc> in one and <abc> in the other).

Table 18 shows the symbols that might appear based on changing values to the node in question. The example illustrated in Figure 141 is shown in the first row. As values in the source documents change, Stylus Studio changes the diff symbol accordingly.


Table 18. Symbols Used to Specify Changes in Source Documents

<i>Symbol</i>	<i>target.xml</i>	<i>source1.xml</i>	<i>source2.xml</i>	<i>source3.xml</i>
	<abc>	<abc>	<xyz>	<123>
	<abc>	<xyz>	<jkl>	<mno>
	<abc>	<xyz>	<xyz>	<xyz>
	<abc>	<xyz>	<abc>	<abc>

Note Changing the target-source pairing, that is, changing the current source document, affects the symbols that are displayed.


How to Diff Multiple Documents


◆ **To diff multiple documents:**

1. Select **Tools > Options > Show Differences > Files** from the Stylus Studio menu. Stylus Studio displays the XML Diff Viewer.
2. In the source document window, click the add button () to add the first source document. Stylus Studio displays the **Open** dialog box.

Tip You can drag and drop a file into the entry field to load the document in the XML Diff Viewer.

3. Navigate to the document you want to load in the XML Diff Viewer.
4. Click **Open**.
5. Repeat [Step 2](#) through [Step 4](#) for additional source documents.

6. Repeat [Step 2](#) through [Step 4](#) for the target document, using the set button for the target document window ().

By default, Stylus Studio runs the diff calculation automatically when you select the target document. If the default **On changes** setting has changed, you need to run the diff calculation manually by clicking the **Calculate diff** button ().

Modifying Default Diff Settings

Default settings for the behavior of the Diff engine and the appearance of diffed documents and folders are on the **Engine** and **Presentation** pages, respectively, of the **Options** dialog box. The **Engine** page, shown here, has settings that determine the conditions under which Stylus Studio runs the diff automatically, which items in a document (comments and text, for example) you want the diff engine to ignore, and settings that allow you to choose diffing algorithm tunings optimized for change description or time, for example.

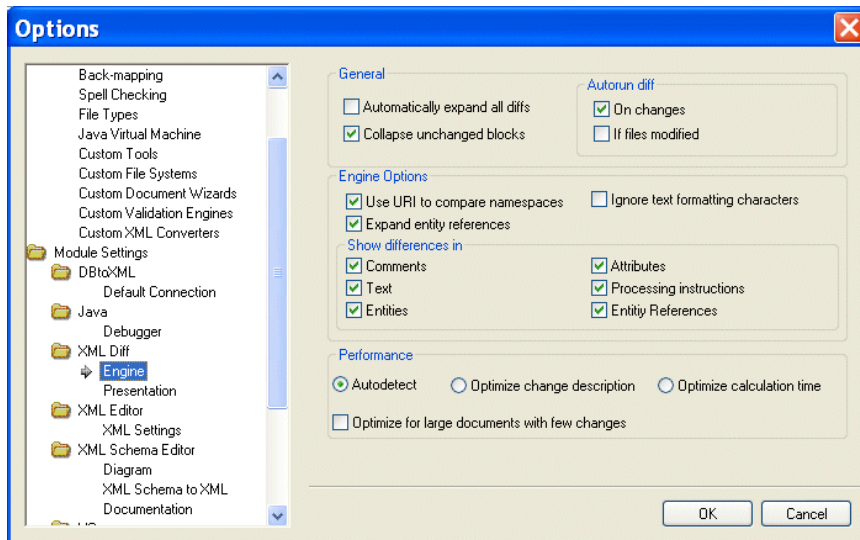


Figure 142. Engine Options Page for XML Diff

Settings on the **Engine** page are reflected in the Diff editor tool bar, and in the **XML Diff** menu, shown here.

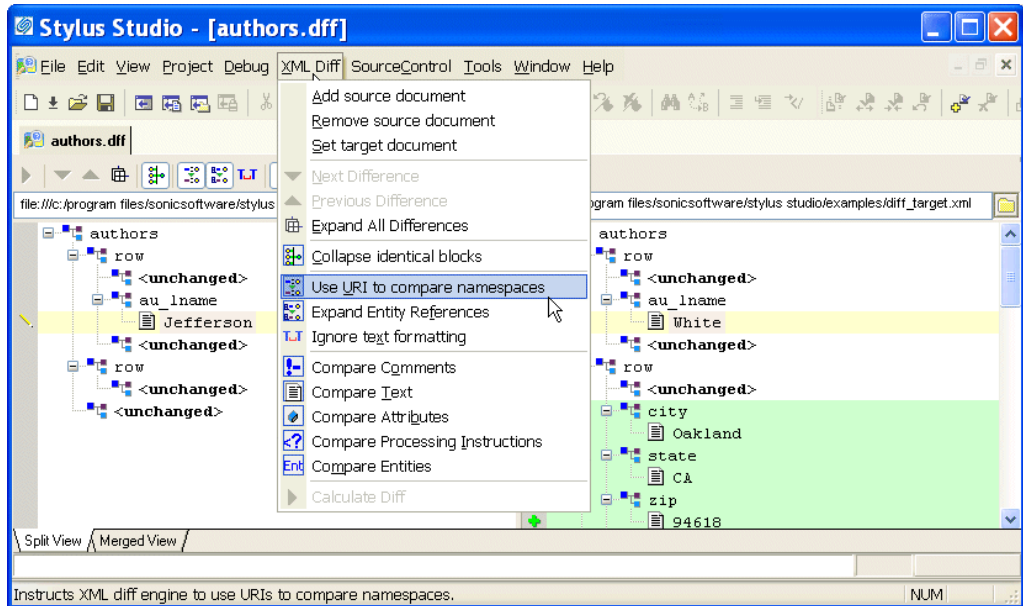


Figure 143. XML Diff Menu

Tip You can override **Engine** page settings using the Diff editor tool bar or the **XML Diff** menu. Overrides do not change the settings on the **Engine** page.

This section covers the following topics:

- “Opening the Options Dialog Box” on page 202
- “Engine Settings” on page 203
- “Presentation Options” on page 205

Opening the Options Dialog Box

◆ **To open the Options dialog box:**

1. Click **Tools > Options** on the Stylus Studio menu.
The **Options** dialog box appears.
2. If necessary, expand the tree for **Module Settings > XML Diff**.

3. Click the page that contains the settings you want to modify.
4. To save a changed setting, click **OK**. Click **Cancel** to revert to close the dialog box and revert to the previous settings.

Engine Settings

This section describes the settings that affect the behavior and performance of the Diff engine.

General

The fields in the **General** group box affect the initial display of diffed documents and the conditions, if any, under which Stylus Studio runs the diff automatically.

- **Automatically expand all diffs** – By default, Stylus Studio collapses the display of the diffed documents. If you select this option, all nodes containing diffs are expanded when the diff is run.
- **Collapse unchanged blocks** – By default, Stylus Studio collapses any block that does not contain any changes to save space in the XML Diff Viewer window. These blocks are displayed as **<unchanged>** in the document tree. You might prefer to have the entire document structure visible, to provide context for changed nodes, for example.
- **Autorun diff** – By default, Stylus Studio runs the diff operation if you make a change to one of the settings on the **Engine** page of the **Options** dialog box. You can also specify that Stylus Studio run the diff operation when the source or target documents change. If you select **If files modified**, Stylus Studio runs the diff operation when you save a source or target file. If neither of these options is selected, you must run the diff manually. See [“When Does the Diff Run?”](#) on page 181 for more information on this topic.

Engine Options

The fields in the **Engine Options** group box affect how Stylus Studio diffs source and target documents.

- **Use URI to compare namespaces** – Controls whether or not URIs are used to compare namespaces in source and target documents.
- **Expand entity references** – Controls whether or not entity references, which in some cases can include files external to the source or target document, are expanded by the Stylus Studio diff engine for purposes of comparing one block with another.

- **Ignore text formatting characters** – Controls whether or not text formatting characters (new line, carriage return, and tab) are ignored when comparing source and target documents. This option is off by default.
- **Show differences in** – Provides granular control of what items in XML documents are diffed. There are separate settings for comments, text, entities, attributes, and processing instructions.

Performance

Diffing large, numerous, or complex documents can be time-consuming. Stylus Studio provides controls that let you choose algorithm tunings that have been optimized for change description or calculation time.

- **Autodetect** – Stylus Studio determines which algorithm tuning to use based on the number, content, complexity, and size of the source and target documents. Stylus Studio first tries to use the tuning that is optimized for change description; if it determines that processing resources are limited, it reverts to the algorithm tuning optimized for speed. This setting is on by default.
- **Optimize change description** – Provides the most economical set of changes possible. This calculation, though it yields the best results, can be costly in terms of time and processing resources.
- **Optimize calculation time** – Provides the set of changes in the shortest time possible.
- **Optimize for large documents with few changes** – Helps speed the diffing of large (greater than 1MB) documents by folding similar blocks before comparing nodes. This setting can be used in conjunction with any of the algorithm tuning settings and is on by default.

Tip The default setting, **Autodetect** and **Optimize for large documents with few changes**, yields the best results when time and processing resources are not considerations.

Presentation Options

Presentation options allow you to modify the settings for the background colors Stylus Studio uses to identify the types of changes detected in diffed documents and folders

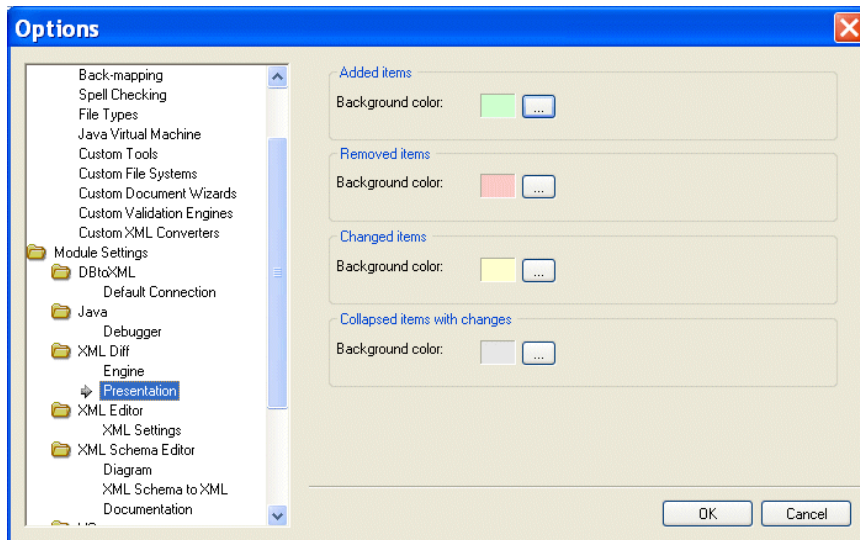


Figure 144. You Can Change Colors Used to Identify Document Changes

You can change the background colors for the following:

- Added items
- Removed items (that is, items that are present in the target, but are not present in the source, for example)
- Changed items (the same element or attribute, but a different name, for example)
- Collapsed items with changes

Running the Diff Tool from the Command Line

In addition to using the Diff tool from the Stylus Studio user interface, Stylus Studio also provides a command line utility, `StylusDiff.exe`. This command line utility allows you to perform many of the same functions, and to use many of the same options, as the graphical Diff tool.

Restrictions

The following restrictions exist for using StylusDiff.exe:

- You cannot use StylusDiff.exe to diff folders
- StylusDiff.exe can diff only one pair of documents at a time

Usage

The StylusDiff.exe utility has the following usage:

```
StylusDiff -source <sourceURI> -target <targetURI>
[-expandPrefixes/collapsePrefixes] [-expandERs/collapseERs]
[-comments/nocomments] [-attributes/noattributes] [-text/notext] [-pi/nopi]
[-er/noer] [-entities/noentities] [-formatting/noformatting] [-fold/nofold]
[-auto/best/fast]
```

[Table 19](#) describes the usage for the StylusDiff command. For a complete description of these and other options that affect the XML Diff engine, see “[Engine Settings](#)” on page 203.

Table 19. StylusXql Command Line Parameters

<i>Parameter</i>	<i>Description</i>
-source <sourceURI>	The path of the XML document you want to use as the source document in the diff. Required.
-target <targetURI>	The path of the XML document you want to use as the target document in the diff. Required.
-output <outputURI>	Saves the differences between the source and target files, if any, to the file you specify. Output files are saved with a .dff extension. Optional.
-expandPrefixes -collapsePrefixes	Whether you want the XML Diff engine to use the URI (-expandPrefixes) or ignore the URI (-collapsePrefixes) when comparing namespaces. The default is -collapsePrefixes.
-expandERs -collapseERs	Whether you want the XML Diff engine to compare entity references (-collapseERs) or values referenced by entity references (-expandERs). The default is -expandERs.

Table 19. StylusXql Command Line Parameters

<i>Parameter</i>	<i>Description</i>
-comments -noComments	Whether you want the XML Diff engine to compare comments (-comments) or to ignore comments (-noComments). The default is -comments.
-attributes -noAttributes	Whether you want the XML Diff engine to compare attributes (-attributes) or to ignore attributes (-noAttributes). The default is -attributes.
-text -noText	Whether you want the XML Diff engine to compare text (-text) or to ignore text (-noText). The default is -text.
-pi -noPI	Whether you want the XML Diff engine to compare processing instructions (-pi) or to ignore processing instructions (-noPI). The default is -pi.
-er -noER	Whether you want the XML Diff engine to compare entity references (-er) or to ignore entity references (-noER). The default is -er.
-entities -noEntities	Whether you want the XML Diff engine to compare entities (-entities) or to ignore entities (-noEntities). The default is -entities.
-formatting -noFormatting	Whether you want the XML Diff engine to compare formatting characters (-formatting) or to ignore formatting characters (-noFormatting) when comparing text nodes. The default is -formatting.
-fold -noFold	Whether you want the XML Diff engine to fold similar blocks before diffing (-foldUnchangedBlocks) or to expand and diff nodes (-diffUnchangedBlocks). The default is -foldUnchangedBlocks.
-auto -best -fast	Controls that let you choose between diffing algorithm tunings that have been optimized for time (-fast) and thoroughness (-best). A third choice, -auto, lets Stylus Studio determine which tuning to use. The default is -auto.

Using Schemas with XML Documents

Stylus Studio allows you to associate one or more schemas with each XML document. A schema can be a DTD or an XML Schema.

There are several ways to associate a schema with an XML document:

- To associate an external schema with a document, ensure that an XML document is active. Then, from the Stylus Studio menu bar, select **XML > Associate XML with Schema**.
- To define an internal DTD, specify it in the XML editor **Text** tab or **Schema** tab.
- To have Stylus Studio generate a schema, in the XML editor, click the **Schema** tab. If the XML document has some contents, Stylus Studio prompts you to indicate whether you want Stylus Studio to generate a schema from the contents. See [“Having Stylus Studio Generate a Schema”](#) on page 209.

This section discusses the following topics:

- [“Associating an External Schema With a Document”](#) on page 208
- [“Having Stylus Studio Generate a Schema”](#) on page 209
- [“Validating XML Documents”](#) on page 209
- [“Updating a Document’s Schema”](#) on page 210
- [“Removing the Association Between a Document and a Schema”](#) on page 210

Associating an External Schema With a Document

- ◆ **To associate an external schema with an XML document:**
 1. Open the XML document you want to associate with a schema. See [“Opening Files in Stylus Studio”](#) on page 93.
 2. From the Stylus Studio menu bar, select **XML > Associate XML with Schema**.
 3. In the **Open** dialog box that appears, navigate to and select the schema you want to associate.
 4. Click **Open**. The selected schema is now associated with the document.

To associate an XML document with an XML Schema, the XML document must contain a root element.

After you associate a schema with a document, you can view a tree representation of the schema in the XML editor window. Click the **Schema** tab. See “[Updating a Document’s Schema](#)” on page 210.

Having Stylus Studio Generate a Schema


In the XML editor, you can click the **Schema** tab to view the schema for your document. If your document does not specify a schema, Stylus Studio displays the **Schema Not Found** dialog box. This dialog box prompts you to indicate whether you want Stylus Studio to create a schema for your document based on its contents.

You can select **Generate XML Schema** or **Generate DTD**. If you select **Generate XML Schema**, you must specify an absolute path for the file that contains the new schema.

If you select **Generate DTD**, you must indicate whether you want the DTD to be internal or external. If it is internal, Stylus Studio inserts it immediately after the XML declaration. If it is external, you must specify or select an absolute path for the file that contains the new DTD.

After you click **OK**, Stylus Studio displays the new schema in the **Schema** tab.

Validating XML Documents

At any time, you can validate your XML document against its schema. Click **Validate Document**  in the window of the document you want to validate.

Stylus Studio displays a message that indicates whether or not your document is valid. If your document does not conform to its schema, Stylus Studio displays a list of error messages that describe the inconsistencies. This list includes line and column numbers that indicate the location of the error. When you click in an XML document, Stylus Studio shows the line and column number in the bottom right corner of the Stylus Studio window.

When Stylus Studio validates a document, it also checks for well-formedness.

Stylus Studio uses font color to indicate valid and invalid element names. Purple fonts indicate valid elements. Orange fonts indicate elements that are not in the schema.


Note Stylus Studio uses Apache's Xerces XML Parser to validate XML documents. Error messages about invalid documents are generated by the Xerces XML Parser. Stylus Studio has no control over the contents of these messages. If you have trouble understanding such a message, try searching the W3C XML Schema Recommendation for the main phrase in the error message.

Updating a Document's Schema

How you update your document's schema depends on whether the schema is internal or external. If the schema is an internal DTD, you can update it in the **Schema** tab of the XML editor.

If the schema is not an internal DTD, you can update it only in the DTD editor or the XML Schema editor. You can, however, view the schema in the **Schema** tab of the XML editor.

When Stylus Studio displays the schema for your document, you can also view the properties for each node in the schema. If the **Properties** window is not already in view, select **View > Properties**. Click on any node in the schema view to see the properties for that node.

To view the text of an external schema or to edit an external schema, you must display it in the DTD editor or the XML Schema editor. To do this, select **XML > Open Associated Schema** from the Stylus Studio menu bar, or click **Open Schema**  in the Stylus Studio tool bar.

Instructions for updating a DTD are in “[Defining Document Type Definitions](#)” on page 603. Instructions for updating an XML Schema are in “[Creating an XML Schema in Stylus Studio](#)” on page 510. If you update a schema in Stylus Studio and that schema is associated with an XML document that is open in Stylus Studio, Stylus Studio refreshes the schema information for the open XML document.

Removing the Association Between a Document and a Schema

To remove the association between a document and an external schema, you must edit the XML document in the **Text** or **Tree** tab. Remove the text or nodes that specify the external schema.

To remove an internal DTD from a document, delete the text or nodes that specify the internal DTD.


Converting XML to Its Canonical Form

By default, Stylus Studio creates XML that conforms to the W3C XML 1.0 recommendation. You can also convert any XML document to its canonical form. When you convert XML to its canonical form, the resulting document conforms to the W3C Canonical XML 1.0 recommendation.

Tip Although you can undo conversion to canonical form, consider using **Save As** to create a copy of the XML document prior to conversion.

◆ **To convert an XML document to its canonical form:**

1. Open the XML document you want to convert to canonical XML.
2. Select **Edit > Make Canonical XML** from the Stylus Studio menu.

Alternative: Click the Make Canonical XML button () on the tool bar.

The XML document is converted to its canonical form.

You can undo this operation (**Edit > Undo**) if necessary.

Querying XML Documents Using XPath


You can use the XML Path Language (XPath) to query XML documents to obtain a subset of the information in that document. You can also query XML Schema and XSLT, provided you open the XSLT using the XML Editor. (You cannot query DTD schema because it is not XML.)

In Stylus Studio, you query XML documents using the XPath Query Editor. To learn more about XPath and how to use the XPath Query Editor, see “[Writing XPath Expressions](#)” on page 629.

Printing XML Documents

You can print the raw XML text view of your document. You cannot print the other views of your document.


◆ **To print a document:**

1. In your XML document, click the **Text** tab.
2. In the Stylus Studio tool bar, click **Print**  or press Ctrl+P.

- ◆ To preview your document before you print it, select **File > Print Preview** from the Stylus Studio menu bar.
- ◆ To specify print options for your document before you print it, select **File > Print Setup** from the Stylus Studio menu bar.

Saving XML Documents

When you save a document, Stylus Studio saves it in the encoding that is specified in the initial XML processing instruction.

- ◆ **To save an XML document:**
 1. Ensure that the window that contains your XML document is the active window.
 2. From the Stylus Studio menu bar, select **File > Save**.
Alternatives: Press Ctrl+S or click **Save**  in the Stylus Studio tool bar.

Options for Saving Documents

The **Application Settings** page of the **Options** dialog box contains two options that affect when and how documents are saved in Stylus Studio. You can choose to have Stylus Studio

- Save modified documents every few minutes. This option is off by default, and has a default setting of 10 minutes.
- Create a backup copy of a document when it is saved.

More About Backup Files

Backup copies are created with a *.bak extension appended to the original document name when saved to the Stylus Studio file system. For example, the backup copy of books.xml would be books.xml.bak. If you are saving to an external file system (such as Raining Data[®] TigerLogic[®] XDMS), the file system manages the backup file name.

Backup files are written to the same file system as the original document. They are not displayed in the **Project** window, and they appear in the **File Explorer** window only if you change the filter to display *.bak files.

Opening a Backup File

You can open a backup file

- From the **File Explorer** window, by double-clicking the file name or selecting **Open** or **Open With** from the file's shortcut menu, for example
- From the **Project** window, by selecting the file and then selecting either
 - **Open Latest Backup** from the file's shortcut menu (right-click to display), or
 - **Project > Open Document's Latest Backup** from the Stylus Studio menu

Chapter 3 **Converting Non-XML Files to XML**



DataDirect XML Converters™ and custom XML conversions are available only in Stylus Studio XML Enterprise Suite.

Stylus Studio uses DataDirect XML Converters™ to convert incoming streams of data from native formats to outgoing streams of XML, and vice-versa. Stylus Studio includes XML Converters for EDI, CSV, binary, and many other file formats. You can also create custom XML converters to convert formats not already supported by DataDirect XML Converters.

This chapter describes how to use DataDirect XML Converters to convert files in Stylus Studio, how to create your own custom XML converters, and how to use files you convert on-the-fly elsewhere in Stylus Studio – as a source for XQuery and XSLT design, for example.



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the Introduction to the Custom XML Conversions module video](#).

You can read about other video demonstrations for the custom XML conversion definition module here: http://www.stylusstudio.com/learn_convert_to_xml.html#converttoxml.

A complete list of all the videos demonstrating Stylus Studio’s features is here: http://www.stylusstudio.com/xml_videos.html.

This chapter covers the following topics:

- “[Introduction](#)” on page 216
- “[DataDirect XML Converters](#)” on page 217
- “[Custom XML Converters](#)” on page 223

- “The Custom XML Conversion Definition Editor” on page 226
- “Parts of an Input File” on page 237
- “Working with Regions” on page 239
- “Working with Fields” on page 247
- “Controlling XML Output” on page 255
- “Creating a Custom XML Conversion Definition” on page 266
- “Using Custom XML Conversion Definitions in Stylus Studio” on page 268
- “Working with EDI Conversions” on page 271
- “The Converter URL Scheme” on page 284
- “Custom XML Conversion Definitions Properties Reference” on page 290

Introduction

When you open or save a file in Stylus Studio, you have the option of converting that file to or from XML using

- DataDirect XML Converters™. A DataDirect XML Converter is a component that allows you to convert non-XML files to XML, and vice versa. DataDirect XML Converters provide support for EDI, CSV, dBase, RTF, and other common file formats.
DataDirect XML Converters are also available as a standalone run-time component on both Java and .NET platforms. DataDirect XML Converters are bundled with Stylus Studio 2007 XML Enterprise Suite.
- A custom XML converter created using the Stylus Studio Custom XML Conversion Definition Editor. Using the Custom XML Conversion Definition Editor, you can create custom XML conversion definitions to handle proprietary file formats or extensions to formats already supported by DataDirect XML Converters.

Accessing Conversion Tools

You can access XML Converters from the Stylus Studio desktop (via the **Open** or **Save** dialog box), through a URL scheme, or by using the DataDirect XML Converters standalone components for Java™ and .NET . To learn more about the DataDirect XML Converters for Java and .NET, see the DataDirect XML Converters documentation at <http://www.xmlconverters.com/doc/>.

Other Ways to Convert Files to XML

In addition to using XML Converters and creating your own custom XML conversions, you can convert files to XML using Stylus Studio document wizards – Stylus Studio document wizards help you convert XML Schema, DTD, and HTML to XML. See “[Using Document Wizards to Create XML](#)” on page 141 for more information.

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.

DataDirect XML Converters in Stylus Studio

Stylus Studio provides access to DataDirect XML Converters:

- Through a graphic user interface (GUI), available when you open and save files
- Through a URL scheme

This chapter describes how to use DataDirect XML Converters in Stylus Studio. To learn about the Java and .NET DataDirect XML Converters standalone components for Java and .NET, see the DataDirect XML Converters documentation at

<http://www.xmlconverters.com/doc/>.

Types of XML Converters

DataDirect provides XML Converters for numerous file formats, as shown in the following table.

Table 20. Types of Files Handled by XML Converters

<i>File Category</i>	<i>File Type</i>	<i>Description</i>
EDI	EDIFACT, X12, IATA, and EANCOM	Automatically detects and parses EDIFACT, X12, IATA and EANCOM EDI message types, with options for custom message types and message extensions to cover proprietary EDI-based formats.
Flat Files	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.
	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.
	CSV	Converter for comma-separated values (CSV) files. Supports multiple encodings and options to tune the quote and escape characters. Supports delimiters besides commas.
	dBase	Support for dBase II, III, III+, IV, and V formats.
	DIF	Data Interchange Format (DIF) is a spreadsheet-based file format. There are also XML Converters for SDI and SYLK.
	DotD	Support for Progress Software's OpenEdge text dump file format.
	JavaProps	Support for Java .properties file format, which are used for program configuration, translation, and data storage.

Table 20. Types of Files Handled by XML Converters

<i>File Category</i>	<i>File Type</i>	<i>Description</i>
	JSON	Uses the algorithms on the JSON.org website to read from XML and write to JSON (JavaScript Object Notation), and vice-versa.
	Line	Reads in text one line at a time, wrapping an element around each line and escaping any embedded & or > or < symbols.
	MBox	Parse the standard mbox file format and even handles multi-part messages.
	Pyx	Support for this line-oriented notation for expressing tree-oriented data.
	RTF	Converts rich-text format (RTF) into XML, and vice versa.
	SDI	Super Data Interchange (SDI) is another popular spreadsheet-based file format. There are also XML Converters for DIF and SYLK.
	SYLK	SYLK (Symbolic Link) is another popular spreadsheet-based file format. There are also XML Converters for DIF and SDI.
	TAB	Tab-separated values format commonly associated with MS Excel spreadsheets.
	WinIni	Converter for Windows .ini configuration files.
	WinWrite	Converter for Microsoft WinWrite files; renders XHTML.

Converting Non-XML Files to XML

As mentioned previously, access to XML Converters is presented in graphic user interface, shown here:

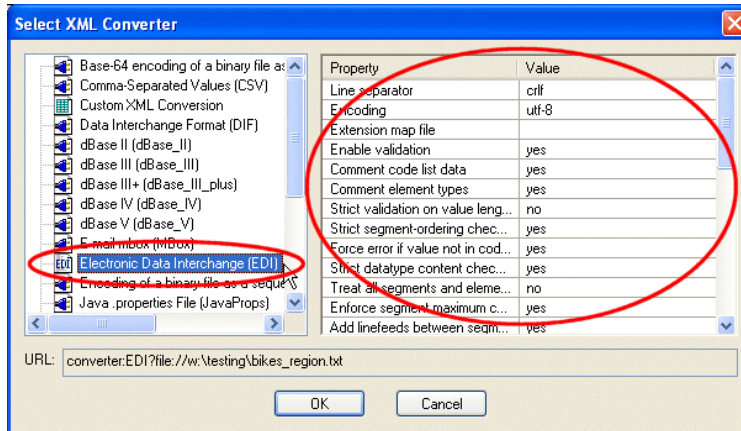


Figure 145. Select XML Converter Dialog Box

To learn more about using XML Converters to convert files to and from XML, see “[Using an XML Converter to Open a Non-XML File as XML](#)” on page 221.

XML Converters Can Be Configured

Each XML Converter has numerous properties that allow you to configure the converter to suit your needs, like those for the EDI XML Converter shown in [Figure 145](#). Some XML Converters, for example, let you specify the line separator character, escape character, root element name, and other aspects of the output format.

See Chapter 4, XML Converters Properties, in the *DataDirect XML Converters User's Guide and Reference* for more information. Documentation for DataDirect XML Converters is available on the DataDirect XML Converters web site:

<http://www.xmlconverters.com/doc/>.

Using an XML Converter to Open a Non-XML File as XML

◆ To open a non-XML file as XML using an XML Converter:

1. Select **File > Open** from the Stylus Studio menu.
Stylus Studio displays the **Open** dialog box.
2. Navigate the file system that contains the file you want to open as XML. If necessary, change the value in the **Files of type** field to filter the files that are displayed.
3. Select the **Open using XML Converters** check box.

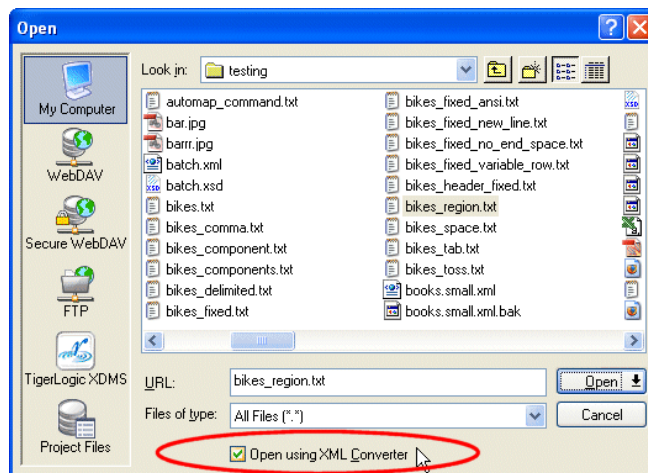


Figure 146. Open Using XML Converters Check Box

4. Click the **Open** button.
Stylus Studio displays the **Select XML Converter** dialog box. (See [Figure 145](#).)
5. Select the XML Converter you want to use to convert your non-XML file to XML.
6. Optionally, change the default values of the conversion properties to be used when converting your file. See Chapter 4, XML Converters Properties, in the *DataDirect XML Converters User's Guide and Reference* if you need help with this step:
<http://www.xmlconverters.com/doc/>.
7. Click **OK**.

Stylus Studio converts the file you specified in [step 2](#) to XML using the DataDirect XML Converter you selected in [step 5](#). The file is displayed in a new instance of the XML Editor.

Saving an XML File in Another Format

DataDirect XML Converters are bi-directional – you can use them to convert native file formats to XML, and vice versa. In order to save an XML file in another format, however, the XML file you are saving needs to have an XML Schema consistent with that expected by the XML Converter. There are a number of ways to achieve this in Stylus Studio:

- Save back a previously converted file. Imagine using the CSV XML Converter to convert `bikes.txt` to `bikes.xml`. You could modify the data in `bikes.xml` and save it back using the CSV XML Converter. Stylus Studio could save the XML file as CSV as long as you made no changes to the document structure (adding no new elements or attributes, for example).
- Use a Document Wizard to create an XML Schema from an EDI message type (EDIFACT, IATA, EANCOM, or X12), and use that XML Schema as the target document in an XQuery or XSLT mapping. The XML document resulting from the XQuery or XSLT transformation based on that XML Schema would conform to the XML Schema expected by the EDI XML Converter.

For more information on Document Wizards, see [“Using Document Wizards to Create XML”](#) on page 141. For more information about working with EDI files as XML, see [“Working with EDI Conversions”](#) on page 271.

Custom XML Converters

In Stylus Studio, you create a custom XML Converter by creating a *custom XML conversion definition* that allows the DataDirect XML Converters engine to convert proprietary files and extensions to standard formats.

You create a custom XML conversion definition by selecting an input file and then using the Custom XML Conversion Definition Editor, shown in [Figure 147](#), to specify the properties you want to use to convert that file (or others that share the same format).

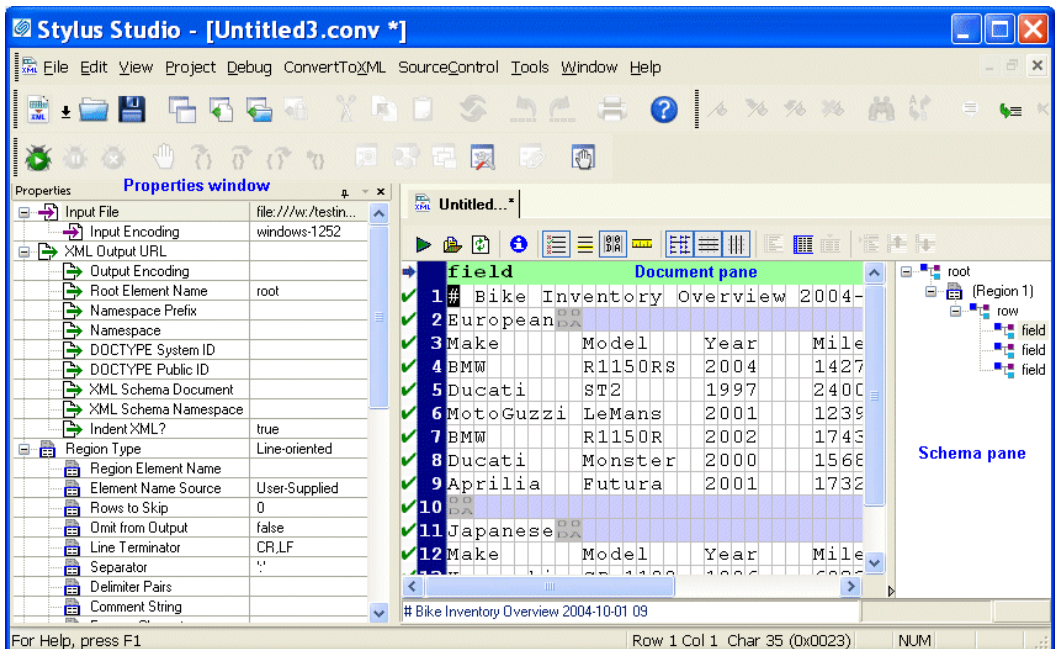


Figure 147. Custom XML Conversion Definition Editor

The Custom XML Conversion Definition Editor displays information read from the input file, as well a number of other properties you can specify (such as whether or not you want element names based on the file's first row, for example) that are used when the file is converted from one format to another.

Creating a Custom XML Conversion Definition

◆ **To create a custom XML Conversion definition:**

1. Select **File > New > Custom XML Conversion** from the Stylus Studio menu to display the **New Custom XML Conversion Definition** dialog box.
2. Select the file you want to convert to XML; optionally, specify the file's encoding and layout, or let Stylus Studio do it for you (the default).
3. Review the default conversion properties displayed in the Custom XML Conversion Definition Editor, and modify as required.
4. Run the custom XML conversion and review the results displayed in the **Preview** window.
5. Save the custom XML conversion (as a .conv file) and, optionally, the XML output for the input file selected in [step 2](#)).

The steps in this process are described in greater detail in the following sections.

Tip Once you have created and saved a custom XML conversion definition, you can use it to open other non-XML files of the same type, just as you would with a built-in DataDirect XML Converter. See [“Using an XML Converter to Open a Non-XML File as XML”](#) on page 221 for more information.

Choosing an Input File

Stylus Studio's XML Converters module makes it easy to define custom XML conversions based on many non-XML file types, including text, binary, and EDI. You can

let Stylus Studio use a set of rules to determine the type, encoding, and layout of the input file, or you can specify these settings manually, as shown in [Figure 148](#).

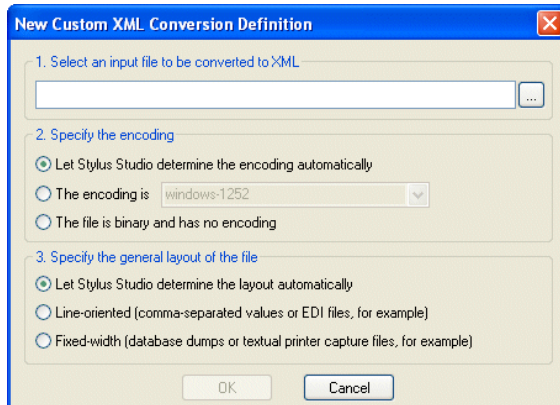


Figure 148. Specify Input File Type Manually or Let Stylus Studio Decide

The input file can be any type. If you plan to use the custom XML conversion definition to convert other non-XML files of this type, the input file should be representative of that broader class of files – files with the same extension (.txt or .edi, for example), encoding, numbers and types of regions, and so on. You can always fine-tune the custom XML conversion definition to accommodate characteristics that are not embodied in the input file, but as a general rule, use a file that is as close to others of its type as possible.

Stylus Studio’s heuristics are also used to determine the field separator character being used (if any), the delimiting character being used (if any), and so on. The assumptions Stylus Studio makes are reflected in the **Properties** window in the Custom XML Conversions Editor once the input file is opened.

The Custom XML Conversion Definition Editor

You use the Custom XML Conversions Editor, shown in Figure 149, to build a custom XML conversion definition. The Custom XML Conversions Editor appears when you create a new custom XML conversion definition, or open an existing one (a .conv file).

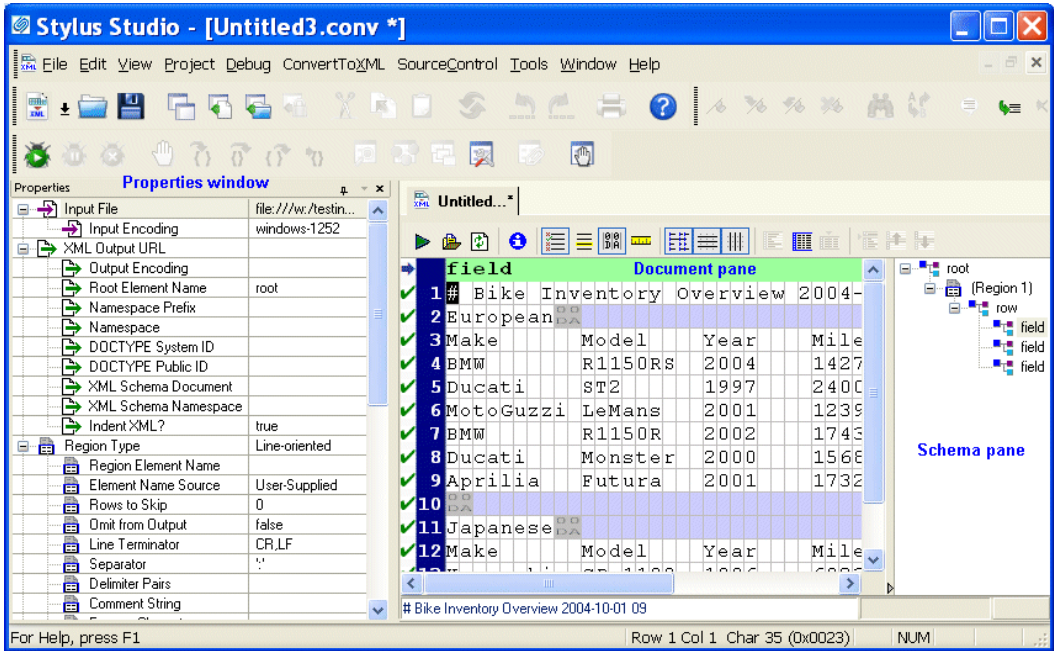


Figure 149. Custom XML Conversions Editor

The input file is displayed in a *document pane*; properties that both describe the existing file (encoding and size, for example) and define the XML output that will be generated when converting this file (root and field element names, and whether or not you want the XML to be indented, for example) are displayed in the **Properties** window. The *schema pane* shows a representation of the XML Schema that will be output for the converted file. Finally, in addition to current row and column position, note that the status bar also shows the Unicode value of the current character.

This section describes the main features of the Custom XML Conversions Editor, including how it interacts with the XML Converters engine. This section covers the following topics:

- [“Document Pane”](#) on page 227
- [“Properties Window”](#) on page 234
- [“Schema Pane”](#) on page 236

Document Pane

The document pane displays the input file’s layout, including spaces, field separators, and control characters. The input file’s appearance in the document pane is determined, in part, by its format.

This section covers the following topics:

- [“Example – .txt Files”](#) on page 227
- [“Display of Delimiting and Control Characters”](#) on page 229
- [“Field Names”](#) on page 230
- [“Document Pane Display Features”](#) on page 231
- [“Moving Around the Document”](#) on page 233

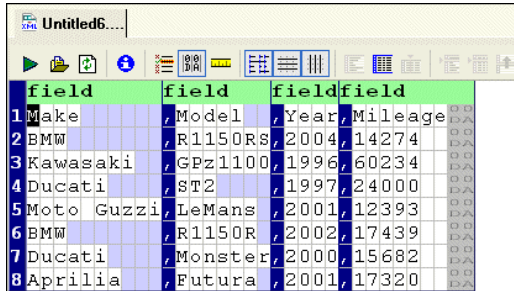
Example – .txt Files

Stylus Studio uses slightly different displays for character-separated and fixed-width .txt files. Consider this file, which uses commas as the field separator:

```
Make,Model,Year,Mileage
BMW,R1150RS,2004,14274
Kawasaki,GPz1100,1996,60234
Ducati,ST2,1997,24000
Moto Guzzi,LeMans,2001,12393
BMW,R1150R,2002,17439
Ducati,Monster,2000,15682
Aprilia,Futura,2001,17320
```


Converting Non-XML Files to XML

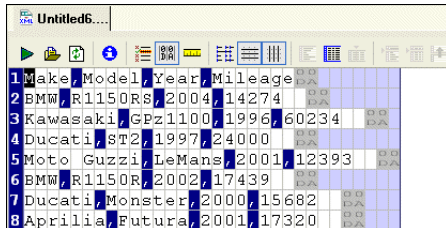
Figure 150 shows how this character-separated input file appears in the Custom XML Conversions Editor's document pane. By default, Stylus Studio aligns columns and fills the empty cells of the shorter rows with a light blue to aid readability:



	field	field	field	field	
1	Make	Model	Year	Mileage	
2	BMW	R1150RS	2004	14274	
3	Kawasaki	GPz1100	1996	60234	
4	Ducati	ST2	1997	24000	
5	Moto Guzzi	LeMans	2001	12393	
6	BMW	R1150R	2002	17439	
7	Ducati	Monster	2000	15682	
8	Aprilia	Futura	2001	17320	

Figure 150. Character-Separated File with Aligned Fields

You can remove these spaces from the display and view the file in its native format by clicking the **Align Fields** button () on the tool bar, or by selecting **CustomXMLConversion > Align Fields** on the menu. This results in the layout shown in Figure 151.



	field	field	field	field	
1	Make	Model	Year	Mileage	
2	BMW	R1150RS	2004	14274	
3	Kawasaki	GPz1100	1996	60234	
4	Ducati	ST2	1997	24000	
5	Moto Guzzi	LeMans	2001	12393	
6	BMW	R1150R	2002	17439	
7	Ducati	Monster	2000	15682	
8	Aprilia	Futura	2001	17320	

Figure 151. Character-Separated File without Aligned Fields

Fixed-width files are displayed in a slightly different fashion. Consider this fixed-width input file:

```
Deep-sea octopus      Bathypolypus arcticus http://www.dal.ca/~ceph/TCP/barctic.html
Blue-ringed octopus  Hapalochlaena lunulatahttp://www.dal.ca/~ceph/TCP/lunulata.html
Caribbean reef octopusOctopus briareus      http://www.dal.ca/~ceph/TCP/obriar.html
Giant octopus        Octopus dofleini      http://www.dal.ca/~ceph/TCP/giant.html
Common octopus       Octopus vulgaris      http://www.dal.ca/~ceph/TCP/Octopusvulgaris.html
Red octopus          Octopus rubescens     http://www.dal.ca/~ceph/TCP/reducto.html
Octopus Salutii     Octopus salutii       http://www.dal.ca/~ceph/TCP/Osalutii.html
Octopus Macropus    Octopus macropus      http://www.dal.ca/~ceph/TCP/Omacropus.html
```



In a fixed-width file, the empty cells represent actual values (spaces) in the input file. In the second row of this input file, for example, there are three spaces between the first and second columns:

field	field	field	field
1 Deep-sea octopus	Bathypolypus arcticus	http://www.dal.ca/~ceph/TCP/barctic.html	
2 Blue-ringed octopus	Hapalochlaena lunulata	http://www.dal.ca/~ceph/TCP/lunulata.html	
3 Caribbean reef octopus	Octopus briareus	http://www.dal.ca/~ceph/TCP/obriar.html	
4 Giant octopus	Octopus dofleini	http://www.dal.ca/~ceph/TCP/giant.html	
5 Common octopus	Octopus vulgaris	http://www.dal.ca/~ceph/TCP/Octopusvulgaris.html	
6 Red octopus	Octopus rubescens	http://www.dal.ca/~ceph/TCP/redocto.html	
7 Octopus Salutii	Octopus salutii	http://www.dal.ca/~ceph/TCP/Osalutii.html	
8 Octopus Macropus	Octopus macropus	http://www.dal.ca/~ceph/TCP/Omacropus.html	

Figure 152. Fixed-Width File

Display of Delimiting and Control Characters

Stylus Studio displays delimiting and control characters in a way that distinguishes them from plain text values.


- Delimiting characters, like the comma used in the example in [Figure 150](#), are displayed with a dark blue background. For files that include sub-fields or arrays (like EDI, for example), the sub-field separator character is shown in a different shade of blue. Sub-sub-fields delimiting characters are shown in a shade of purple.
- Control characters (line feeds and carriage returns, for example) are shown using their abbreviated ASCII value. A carriage return (0x0D) line feed (0x0A) is shown as , for example. ASCII abbreviations are aligned vertically, to preserve space, as shown in this representation of the ASCII value for tab (0x09): .

Stylus Studio understands all Unicode characters. When editing **Line-Oriented Region** and **Field Name** values in the **Properties** window, you can enter mnemonic values for the C1 and C0 control characters in the following ranges:

- C0 control characters with a value from $\geq 0x00$ to $\leq 0x1F$
- C1 control characters with a value from $\geq 0x80$ to $\leq 0x9F$

For example, you could enter TAB or HT in the **Field Separator** field in the **Properties** window, and Stylus Studio would correctly interpret that value. For a list of commonly used control characters, see [“Specifying Control Characters”](#) on page 320.

- Characters that are discarded from output (like line terminators such as CR and LF and comment lines) are displayed against a gray background.

You can hide control characters by clicking the **Toggle Control Characters** button () on the tool bar, or by selecting **ConvertToXML > Toggle Control Characters** from the menu.

Field Names

User-defined field names – values that Stylus Studio uses to create the element names in converted XML – are displayed in green, as shown here:

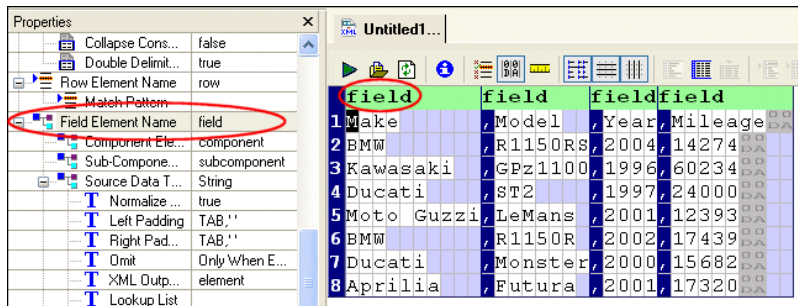


Figure 153. User-defined Field Names are Shown in Green

You can edit these names

- In-place, by double-clicking the field name
- In the **Field Element Name** field of the **Properties** window

If the field names are taken from a row within the file itself, Stylus Studio displays a blue arrow in the document pane margin to indicate this.

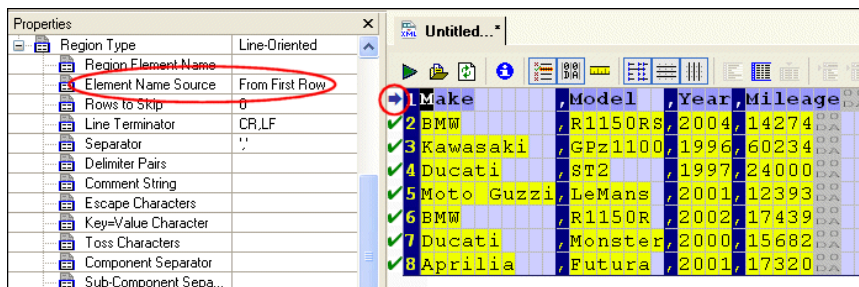


Figure 154. Blue Arrow Indicates Field Names Taken from the File

See “[Naming Fields](#)” on page 247 to learn more about naming fields for XML output by custom XML conversion definitions.

Document Pane Display Features

In addition to aligning fields in character-delimited files, the Custom XML Conversions Editor's document pane has several other features that aid readability.

Ruler

You can display a ruler that identifies each column:

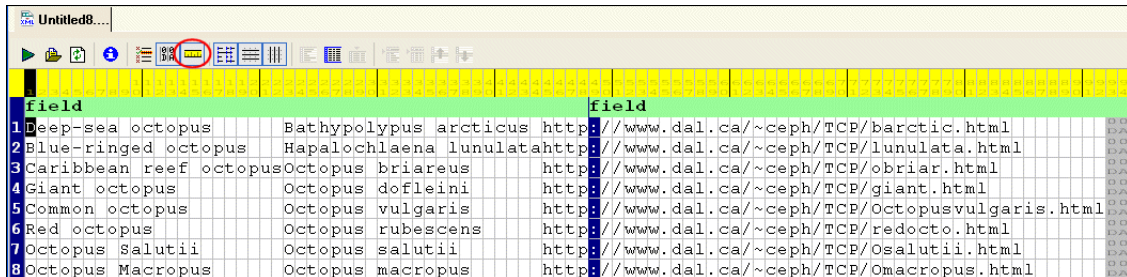


Figure 155. Ruler Helps Identify Columns

To display the ruler, click the **Toggle Ruler** button () on the tool bar, or select **CustomXMLConversion > Toggle Ruler** from the menu.

Displaying Pattern Matches

You can define match patterns using regular expressions to control which rows are converted to XML and, optionally, the name to use for these rows. You can highlight rows that match the patterns that you have defined, as shown here:

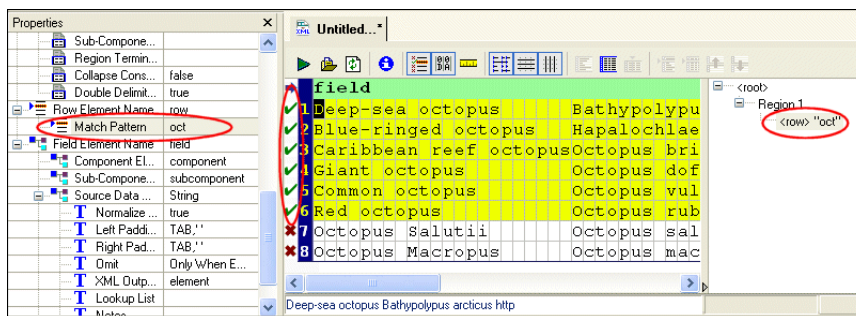


Figure 156. Matching Rows Are Displayed in Yellow

To highlight matching rows, click the **Highlight Matching Rows** button () on the tool bar, or select **CustomXMLConversion > Highlight Matching Rows** from the menu.

Matching rows are displayed in light yellow, with a green check in the pane's margin. A red X identifies rows that do not match the current pattern. Gray squares identify rows that match a pattern other than the pattern defined for the row that currently has focus. See [“Specifying Multiple Match Patterns”](#) on page 259 for more information on this feature.

Tip Only rows that match the same pattern that the current row matches are highlighted. Also, tooltips appear when you hover the pointer over the match symbols. These tooltips display the pattern that the row matches.

See [“Pattern Matching”](#) on page 257 to learn more about using regular expressions to define match patterns.

Grid Lines

The document pane displays both vertical and horizontal lines by default; you can hide/show them independently. In the example shown in [Figure 157](#), horizontal lines are hidden from the display:

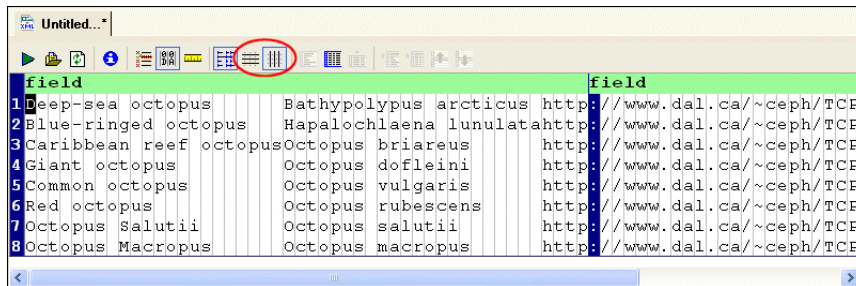


Figure 157. You Can Hide Horizontal and Vertical Grid Lines

To hide horizontal and vertical grid lines, click the **Toggle Horizontal Grid Lines** (☰) and/or **Toggle Vertical Grid Lines** (⏏) buttons on the tool bar, or select **ConvertToXML > Toggle Horizontal Grid Lines** and/or **Toggle Vertical Grid Lines** from the menu.

Tip Hiding horizontal lines while displaying the ruler is an effective way to quickly scan columns.

Fonts

By default, the input document is displayed using the Courier New font in 12pt. You can change the display font to suit your personal preference using the **Edit > Change Font** and **Edit > Font Size** menus.

Moving Around the Document

You can move the cursor around the document using

- The Space bar on your keyboard
- The directional arrows and Page Up, Page Down, Home, and End keys on your keyboard
- Your mouse (click on the character on which you want to place the cursor)
- The **Go To** dialog box

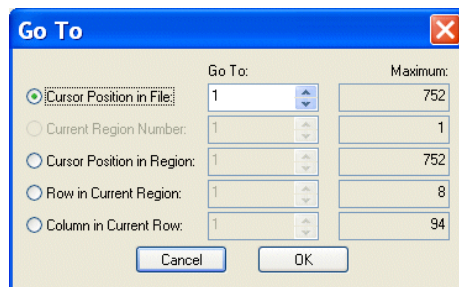


Figure 158. Use the **Go To** Dialog Box to Navigate the Document

Using Go To

You use the **Go To** dialog box to jump to a specific location in the file you are using to create your custom XML conversion definition. You can use it to move the cursor to a specific

- Position in the file
- Region
- Position or row within a region
- Column within the current row

When you first display the **Go To** dialog box, values in the **Go To** fields reflect the cursor's current location within the file. The values in the **Maximum** fields display the maximum values for each category (file size, number of regions, and so on) for the portion of the file read into the Custom XML Conversions Editor Editor by Stylus Studio.

- ◆ To display the Go To dialog box, select Edit > Go To from the menu.

Properties Window

The **Properties** window, like the one shown in [Figure 159](#), displays information about the input file, as well as settings that Stylus Studio will use convert files to XML.

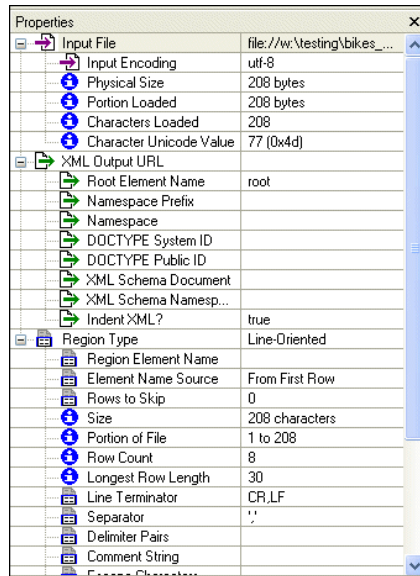









Figure 159. Properties Window for a .txt File


Information in the **Properties** window includes

- Information read or inferred from the input file when it is first opened in the Custom XML Conversions Editor. Examples include the file name, file size, and number of characters that were read. Some values, such as the type of encoding, can be edited. Informational fields that cannot be edited are identified with a blue circle: .
- Values you want the XML Converters engine to use when converting this input file and other files of this type to XML. Output properties include the root element name, the namespace and namespace prefix, and the field element name. These fields are identified with a green arrow over a document icon: .

How Properties are Organized

Properties displayed in the **Properties** window are organized in the following categories:

- **Input File** – read-only information read or inferred from the input file, and editable properties that affect XML output. These properties affect the file as a whole when it is converted to XML. Input file properties are identified by this icon: .
- **XML Output URL** – properties that affect the XML document created by the custom XML conversion definition, including the name you want to use for the root element, and whether or not you want to indent the XML. Output URL properties are identified by this icon: .
- **Region Type** – read-only information inferred from the input file, and editable properties that affect XML output. Examples include line terminating and escape characters. These properties affect a contiguous portion of the file (that is, a given line-oriented or fixed-width region) when it is converted to XML. Region properties are identified by this icon: .
- **Row Element Name** – properties that affect which rows of the input file are output to XML and how they are output, including the name you want to use for the row. Row properties are identified with this icon: .
- **Field Element Name** – read-only information read or inferred from the input file, and editable properties that affect XML output. These properties affect only fields in a given region of the file when it is converted to XML. Field properties are identified by this icon: .

Note Informational properties, that is, properties that do not affect XML output, are displayed with the following icon: . These properties are displayed when you click the **Toggle Informational Properties** button.

Properties for Fixed-Width and Line-Oriented Input Files

Fixed-width and line-oriented input files have different properties – line-oriented properties include the line terminator and field separator characters, and fixed-width files have a row length, for example. See [“Custom XML Conversion Definitions Properties Reference”](#) on page 290 to learn more about individual properties.

Schema Pane

The schema pane displays a representation of the XML Schema for the XML document that will be output when the input file is converted to XML.

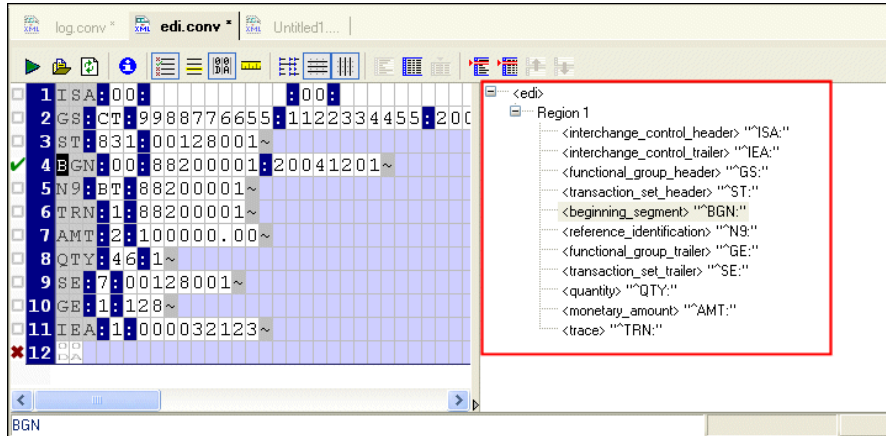


Figure 160. Schema Pane Shows Output Schema Representation

You can double-click on a row element to display the **Set Node and Match Pattern** dialog box, shown in [Figure 161](#). This functionality provides an alternative to editing the row name and specifying a match pattern in the **Properties** window. (To learn more about patterns and how to use them in your custom XML conversion definitions, see [“Pattern Matching”](#) on page 257.)

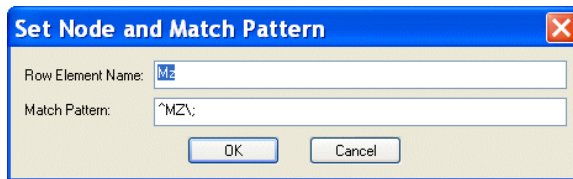


Figure 161. Set Node and Match Pattern Dialog Box

You can also edit schema node names directly in the schema pane – just click twice to place the node name in edit mode.

See [“Rows”](#) on page 238 to learn more about specifying conversion properties for rows.

Parts of an Input File

Input files displayed in the Custom XML Conversions Editor’s document pane consist of regions, rows, and fields. Each section has its own set of properties. Some, like **Region Type**, are read or inferred from the file; others, like **Region Element Name**, are values you provide that affect the XML output.

This section covers the following topics:

- “Regions” on page 237
- “Rows” on page 238
- “Fields” on page 239

Regions

A *region* is the largest organizational component in an input file. Regions are interpreted by Stylus Studio when the input file is first read into the Custom XML Conversions Editor. You can use the editor to define your own, as well.

An input file can contain one or more regions; every input file has at least one region that starts at offset 0. Multiple regions are common in binary files, which often contain a fixed-size header and then one or more records containing the actual data. Regions are fixed in size and cannot repeat.

In the Custom XML Conversions Editor, regions are numbered, starting with 1, followed by the row number. For example, in an input file with two regions, you might see rows labeled as follows: 1:1, 2:1, 2:2, 2:3, and so on, as shown here:

	fieldfield	fieldfield	fieldfield	fieldfield
1:1	Make	Model	Year	Mileage
2:1	BMW	R1150RS	2004	14274
2:2	Kawasaki	GPz1100	1996	60234
2:3	Ducati	ST2	1997	24000
2:4	Moto Guzzi	LeMans	2001	12393
2:5	BMW	R1150R	2002	17439
2:6	Ducati	Monster	2000	15682
2:7	Aprilia	Futura	2001	17320

Figure 162. Rows in Regions Are Numbered Independently

Region Types

Regions can be fixed-width or line-oriented. You can also set the **Region Type** property to **No-output**. Regions that are marked as No-output are grayed out in the Custom XML Conversions Editor, and they are not converted to XML.

Managing Regions

Stylus Studio provides tools that let you create new regions, and join one region with another. You can also change a region's type, define different line terminators across regions, and mark a region so that it is excluded from output. For information on these and other topics, see [“Working with Regions”](#) on page 239.

Rows

A *row* is equivalent to a record, line, or tuple in the input file; a row is made up of fields. An example of a row is an employee record; examples of fields in an employee record include `employee_id`, `last_name`, `first_name`, and so on.

Every region can have one or more rows. (A region cannot be empty.) In addition, a region can have multiple row types. Rows are selected for conversion to XML based on the match patterns expressed in the **Match Pattern** field of the **Properties** window. See [“Omitting Regions and Fields, and Rows”](#) on page 256 for more information.

Rows in a fixed-width region have the same width as the region itself; fields within each row are defined by a fixed number of columns. Stylus Studio uses a default value of 80 characters for row length in fixed-width regions, but you can adjust this as required from within the Custom XML Conversions Editor. See [“Adjusting Fixed-Width Regions”](#) on page 242 for more information.

In a fixed-width region, you can

- Explicitly specify the fields within a row
- Adjust the size of the fields you specify

In a line-oriented region, fields are separated by a separator character or string. These characters are inferred by Stylus Studio when it first reads the file, but you can change these and other characters if needed.

See [“Working with Fields”](#) on page 247 for more information.

Fields

A *field* is one or more columns in a row that contains data. Each different row type has its own independent set of fields. An example of a field in an employee record is `employee_id`.

Stylus Studio supports many data types (string, Boolean, number, date, time, and so on) and recognizes many different input formats (like different date formats, for example). Properties vary based on data type – the **Base** property, for example, is applicable only to Number data types.

You can define your own fields in fixed-width input files. See [“Defining Fields”](#) on page 250 for more information.

Component and Sub-Component Fields

Some file formats, including many EDI dialects, allow fields to be subdivided into arrays, sub-fields, or composite fields. Collectively, these fields are referred to as *component fields* in custom XML conversion definitions, and they are fully supported, both in terms of recognition and output. You can name the container element using the **Component Element Name** and **Sub-Component Element Name** properties.

Working with Regions

This section describes some of the features you can use to work with input file regions. It covers the following topics:

- [“Converting the Region Type”](#) on page 240
- [“Adjusting Fixed-Width Regions”](#) on page 242
- [“Defining and Joining Regions”](#) on page 244
- [“Controlling Region Output”](#) on page 246

Converting the Region Type

The **Region Type** field in the **Properties** window displays information about the type of region Stylus Studio inferred when the file was first read. Its value is either **Fixed-width**, **Line-oriented**, or **No-output**.

Tip Information about **No-output** regions is not displayed in the **Properties** window.

Regions with CR/LF control characters are interpreted as line-oriented regions. There might be occasions, however, when you want to change the region type from line-oriented to fixed-width, or vice versa. This section describes the tools you can use to change a region from one type to another.

Consider the following file fixed-width file:

Make	Model	Year	Mileage
BMW	R1150RS	2004	14274
Kawasaki	GPz1100	1996	60234
Ducati	ST2	1997	24000
MotoGuzzi	LeMans	2001	12393
BMW	R1150R	2002	17439
Ducati	Monster	2000	15682
Aprilia	Futura	2001	17320

It is a simple .txt file with fields (Make, Model, and so on) that have been created using spaces. Each new row was created using the Enter key in the text editor, resulting in CR/LF control characters at the end of each line that cause Stylus Studio to interpret the file as a single line-oriented region, like this:

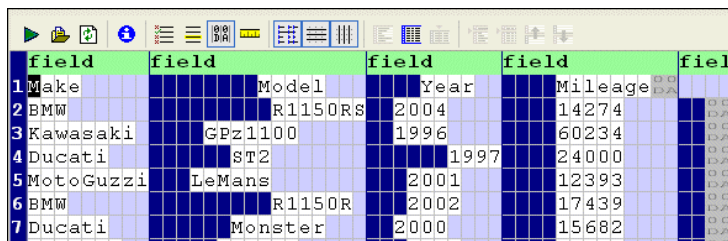


Figure 163. Fields are Aligned by Default

For display purposes, we can remove the spaces Stylus Studio has inserted for readability (the cells with the light blue shading) by clicking the **Align Fields** () button. This

results in a *display* that resembles the source (Figure 164), but Stylus Studio still considers the region to be line-oriented.

1	Make	Model	Year	Mileage
2	BMW	R1150RS	2004	14274
3	Kawasaki	GPz1100	1996	60234
4	Ducati	ST2	1997	24000
5	MotoGuzzi	LeMans	2001	12393
6	BMW	R1150R	2002	17439
7	Ducati	Monster	2000	15682
8	Aprilia	Futura	2001	17320

Figure 164. Turning Off Align Fields Can Aid Readability

When you convert a line-oriented region to a fixed-width region, Stylus Studio removes spaces it added for readability and depicts only the spaces in the original input file used to create the fields and the field values themselves, as show in Figure 165.

field
1 Make Model Year Mileage BMW R1150RS 2004 14274 Kawasaki GPz1100 1996 60234 Ducati ST2 1997 24000 MotoGuzzi LeMans 2001 12393 BMW R1150R 2002 17439 Ducati Monster 2000 15682 Aprilia Futura 2001 17320
4



Figure 165. Line-Oriented Regions Converted to Fixed-Width

By default, Stylus Studio displays fixed-width files using an 80-character row. This accounts for the input files appearance when it is first displayed as a fixed-width file – if you scan the document, you can see that all of the file’s original information, including the CR/LF control characters has been retained, but that the formatting differs – the original input file had eight rows; now it has four rows of 80 characters.

Tip You can adjust the width of fixed-width regions. See “[Adjusting Fixed-Width Regions](#)” on page 242.

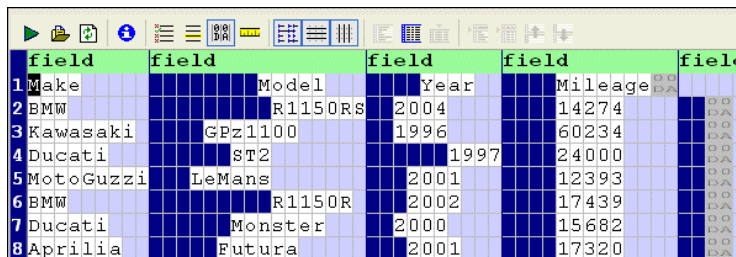
How to Convert a Region Type

◆ **To convert a region type:**

1. Place the cursor anywhere in the region you want to change.
2. Click the **Convert to Fixed-Width Region** () or **Convert to Line-Oriented Region** () button. These actions are also accessible from the **CustomXMLConversion** menu and the shortcut menu in the Custom XML Conversions Editor.
3. If you have converted a line-oriented region to a fixed-width region, adjust the row length as needed. See [“Adjusting Fixed-Width Regions”](#) on page 242.

Adjusting Fixed-Width Regions

If you [specify the layout](#) of the file you are [converting](#) as fixed-width, Stylus Studio uses a default row length of 80 characters. (If you let Stylus Studio determine the file layout, Stylus Studio will attempt to determine record length based on the line terminating character, if any.) You might need to adjust the row length of a region if your input file uses a different row length, or when converting a line-oriented region, like the one shown in [Figure 166](#), to fixed-width.



	field	field	field	field	field
1	Make	Model	Year	Mileage	
2	BMW	R1150RS	2004	14274	
3	Kawasaki	GPz1100	1996	60234	
4	Ducati	ST2	1997	24000	
5	MotoGuzzi	LeMans	2001	12393	
6	BMW	R1150R	2002	17439	
7	Ducati	Monster	2000	15682	
8	Aprilia	Futura	2001	17320	

Figure 166. Line-Oriented Region

There are several ways to specify the row length for a fixed-width region:

- Using the **Row Length** property – simply change the default value, 80, to the value that is appropriate for the current region and press Enter.
- Dragging the document pane to the left or right – move the pointer to the right border of the document pane. When it changes shape, press and hold mouse button 1 and drag the right border of the grid until the input file’s fields are aligned.
- Holding the Shift key and pressing the right arrow (to add width) or the left arrow (to decrease width).

Each of these methods lets you explicitly set the row length. Alternatively, you can specify a Line Terminator character manually, as shown in [Figure 167](#).

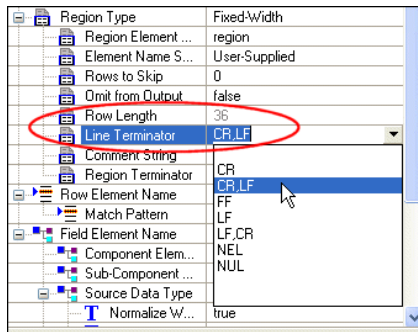


Figure 167. Manually Setting the Line Terminator Character

Specifying a Line Terminator character means that the rows in the region can be of variable length, based on the where the specified Line Terminator character occurs in the record.

Tip When you specify a Line Terminator character for a fixed-width region, the value shown in the **Row Length** property represents the value of the longest row in the region.

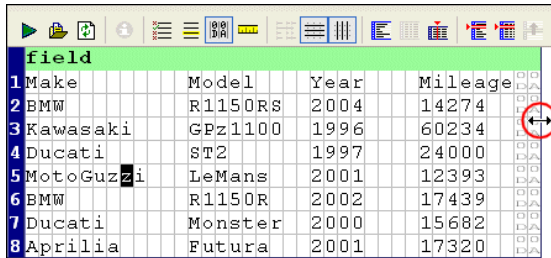
Example

After converting the line-oriented region shown in [Figure 166](#) to fixed-width, it looks like this:

1	Make	Model	Year	Mileage	BMW	R1150RS	2004	14274	Kawa
2	Saki	GPz1100	1996	60234	Ducati	st2	1997	24000	MotoGuzz
3	i	LeMans	2001	12393	BMW	R1150R	2002	17439	Ducati
4	Monster	2000	15682	Aprilia	Futura	2001	17320		

Figure 168. Line-Oriented Region Converted to Fixed-Width

Figure 166 shows the same fixed-width file after it has been resized by dragging the document pane.



field	Make	Model	Year	Mileage	
1	BMW	R1150RS	2004	14274	
3	Kawasaki	GPz1100	1996	60234	
4	Ducati	ST2	1997	24000	
5	MotoGuzzi	LeMans	2001	12393	
6	BMW	R1150R	2002	17439	
7	Ducati	Monster	2000	15682	
8	Aprilia	Futura	2001	17320	

Figure 169. Resized Fixed-Width Region

Defining and Joining Regions

An input file can contain any number of regions; fixed-width and line-oriented regions can exist in the same file. The Custom XML Conversions Editor provides tools that allow you to define new regions and join existing ones.

This section covers the following topics:

- “Defining a Region” on page 244
- “Joining Regions” on page 246

Defining a Region

When you define a region in an input file, Stylus Studio splits the region at the current cursor location. The new region starts with the character on which the cursor resided when the region was defined, but it can be of either type – fixed-width or line-oriented – regardless of the type of the original region.

Consider the following input file:

```
# Bike Inventory Overview 2004-10-01 09:00:07EDT
Make,Model,Year,Mileage
BMW,R1150RS,2004,14274
Kawasaki,GPz1100,1996,60234
Ducati,ST2,1997,24000
Moto Guzzi,LeMans,2001,12393
BMW,R1150R,2002,17439
Ducati,Monster,2000,15682
Aprilia,Futura,2001,17320
```

By default, Stylus Studio reads this as a file with a single region. You might decide you want your XML to distinguish headers from actual records and treat the two accordingly (not generating headers as XML, for example).



When you define a new region, the Custom XML Conversions Editor renumbers all the rows, using a region:row number format. In addition, each region is displayed with its own field name row, which is displayed in light green with the default field element name, field, as shown in [Figure 170](#).

field	field	field	field
1:1	# Bike Inventory Overview	2004-10-01	09:00:07EDT
field	field	field	field
2:1	Make	Model	Year Mileage
2:2	BMW	R1150RS	2004 14274
2:3	Kawasaki	GPz1100	1996 60234
2:4	Ducati	ST2	1997 24000
2:5	Moto Guzzi	LeMans	2001 12393
2:6	BMW	R1150R	2002 17439
2:7	Ducati	Monster	2000 15682
2:8	Aprilia	Futura	2001 17320

Figure 170. Regions Are Numbered and Colored Differently

Field and row values are independent across regions. For example, the <row> element might be <reg1>, <reg2>, and so on for each of the regions in an input file.

◆ **To define a region:**

1. Place the cursor in the document pane on the character with which you want to start the new region.
2. Click the **Start New Line-Oriented Region Here** () or **Start New Fixed-Width Region Here** () button. These actions are also accessible from the **CustomXMLConversion** menu and the shortcut menu in the Custom XML Conversions Editor.

Stylus Studio defines the new region and renumbers existing regions accordingly.

Joining Regions



You can join regions that you define as well as regions that Stylus Studio interpreted when it first read the input file. You can join the current region to either adjacent region – the previous region, or the next region.

The region type after the join operation depends on whether you are joining with the previous region or the next region. The region you are joining assumes the type of the region to which it is being joined.

Table 21. Region Type After Joining Regions

<i>Region Joined With</i>	<i>Resulting Region Type</i>
Next	The region you are using to perform the join
Previous	The region to which you are joining

◆ **To join a region:**

1. Place the cursor anywhere in the region you want to join with another region.
2. Click the **Join with Previous Region** () or **Join with Next Region** () button. These actions are also accessible from the **CustomXMLConversion** menu and the shortcut menu in the Custom XML Conversions Editor.

Stylus Studio joins the region you specified in [step 1](#) with the adjacent region.

Controlling Region Output

By default, Stylus Studio generates output for all fixed-width and line-oriented regions. **No-output** regions are never converted to XML. In addition to pattern matching, which controls whether or not individual rows in a region are output based on a pattern you define, you can omit entire regions from output by selecting **Yes** from the **Omit from Output** drop-down list in the **Region Type** section of the **Properties** window.

Working with Fields

This section describes some of the features you can use to work with input file fields. It covers the following topics:

- “Naming Fields” on page 247
- “Defining Fields” on page 250
- “Component and Sub-Component Fields” on page 253

Naming Fields

Every field in an input file – including fields in the same region and row – can have its own field element name. All field element names (<field> is the default) include the namespace prefix in the XML output if one was specified.

Field names are determined by two properties – **Element Name Source** in the **Region Type** properties, and **Field Element Name**, as shown in [Figure 171](#).

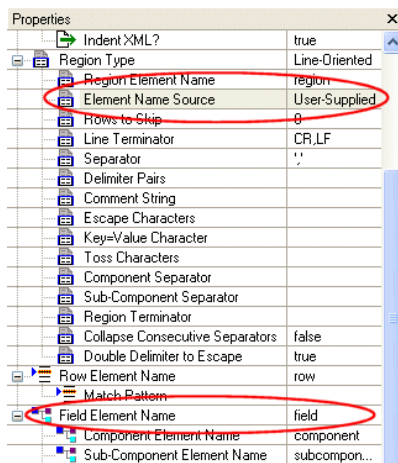


Figure 171. Sources for Field Names in XML Output

The **Element Name Source** indicates the origin of the field name used in the XML output when converting the input file. The **Field Element Name** property specifies the actual value used to name the <field> element.

Using the Element Name Source Property

There are three values for the **Element Name Source** property:

- **User-Supplied** – Specifies that you will supply names for the <field> element. You can do this by editing the **Field Element Name** property, or by double-clicking the field element name in the document pane to edit the field name directly in the document pane.

The default value of the **Field Element Name** property is `field`. If you use other values for the **Element Name Source** property, Stylus Studio provides the values for the **Field Element Name** property.

User-Supplied is the default setting for the **Element Name Source** property.

- **From First Row** – You can use this setting to take <field> element names from the first row in the region. If you have used the **Rows to Skip** property to skip rows in a region, the first available region is used to supply the <field> element names.

Consider the following input file:

```
Make:Model:Year
BMW:R1150RS:2004
MZ:Scorpion:1995
Ducati:ST2:1997
```

If you set **Element Name Source** to **From First Row**, the XML output uses `Make`, `Model`, and `Year` for the <field> element names, as shown here:

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <row>
    <Make>BMW</Make>
    <Model>R1150RS</Model>
    <Year>2004</Year>
  </row>
  <row>
    <Make>MZ</Make>
    <Model>Scorpion</Model>
    <Year>1995</Year>
  </row>
  <row>
    <Make>Ducati</Make>
    <Model>ST2</Model>
    <Year>1997</Year>
  </row>
</root>
```

You can specify *any* row as the source for field names using the **Get Field Names from This Row** from the row's shortcut menu.

- **WS-EDI Standard** – This setting allows rows and fields to be named based on the WS-EDI Standard level 0. See <http://www.ws-edi.org> for more information on this standard.

More About Using Rows for Field Names

When you use an existing row as the source for field names in the XML output, Stylus Studio changes the display of that row in the document pane to a darker blue to indicate this, as shown here:

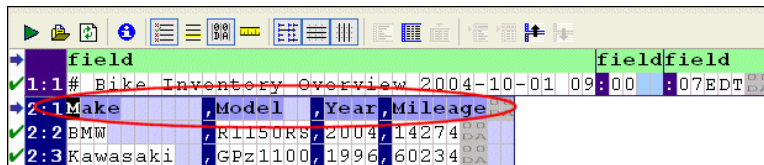


Figure 172. Using a Row for Field Names

In addition, preceding rows in that region, if any, are grayed out, and the value of the **Rows to Skip** field in the **Region Type** properties changes to reflect this.

In the event that the first row has fewer names than there are fields in one or more subsequent lines in the file, Stylus Studio names the extra fields `<fieldn>`, where *n* is the field number relative to other fields in the row. Also, if the values in the row are not valid XML identifiers, they are converted using the following rules:

- Whitespace and nulls are trimmed from both ends
- SQL/XML rules are used, except that underscores (“_”) are not converted to `_x005F_` symbols
- Beyond these exceptions, strict rules are used. See section 9.1 (page 91) of <http://www.sqlx.org/SQL-XML-documents/5FCD-14-XML-2004-07.pdf>.

How to Name Fields

◆ To provide user-supplied field names:

1. Display the **Properties** window if it is not already displayed (click **View > Properties** on the Stylus Studio menu).
2. Place the cursor anywhere in the field you want to name.
The **Field Element Name** property displays the current value for the field.
3. Type the new name in the **Field Element Name** property and press Enter.

Alternative:

1. Double-click the **field** name in the document pane.
The field name field becomes editable.

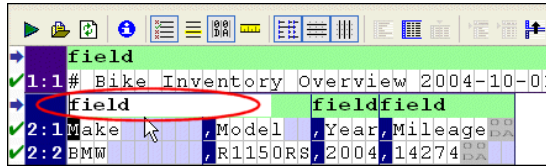


Figure 173. You Can Edit the Field Name Directly in the Document Pane

2. Type a new value for **field** and press Enter.
- ◆ **To specify alternate sources for field names:**
 1. Display the **Properties** window if it is not already displayed (click **View > Properties** on the Stylus Studio menu).
 2. Select the field name source you want to use from the **Element Name Source** drop-down list.
 3. Press Enter.

Defining Fields

You can define fields in any region in a fixed-width input file, as shown in [Figure 174](#). Once you have defined a field, you can change its size by simply dragging it to any column in the grid.

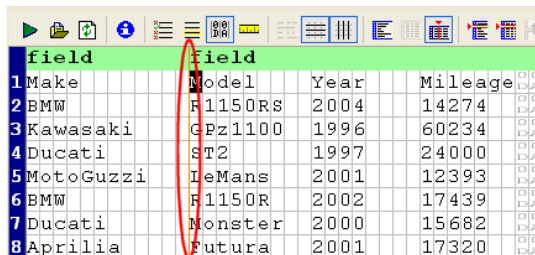


Figure 174. Line Identifying a Field in a Fixed-Width Input File

Each field you define is treated as a separate element in the XML output by the custom XML conversion definition. The input file shown in [Figure 174](#), for example, would result

in XML with two `<field>` elements, one consisting of the make of motorcycle, and one consisting of the model, year, and mileage. You can use the field feature to exercise control over the XML – defining separate fields for make, model, year, and mileage, for example.

Consider the following input file:

Make	Model	Year	Mileage
BMW	R1150RS	2004	14274
Kawasaki	GPz1100	1996	60234
Ducati	ST2	1997	24000
MotoGuzzi	LeMans	2001	12393
BMW	R1150R	2002	17439
Ducati	Monster	2000	15682
Aprilia	Futura	2001	17320

By default, each row is considered to have a single field, containing Make, Model, Year, and Mileage, resulting in XML output like this:

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <row>
    <field>Make          Model    Year    Mileage
  </field>
  </row>
  <row>
    <field>BMW          R1150RS  2004    14274
  </field>
  </row>
  ...
```

Converting Non-XML Files to XML

If you specify fields for Model, Year, and Mileage, the XML output by the custom XML conversion definition looks like this:


```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <row>
    <field>Make</field>
    <field>Model</field>
    <field>Year</field>
    <field>Mileage
  </field>
  </row>
  <row>
    <field>BMW</field>
    <field>R1150RS</field>
    <field>2004</field>
    <field>14274
  </field>
  </row>
  ...

```

Neither approach is always correct, but this feature gives you the ability to define the type of XML output that is appropriate for your use.


Tip Of course, in this example, the next logical step might be to use the first row (Make, Model, Year, and Mileage) as the field names as described in [“Naming Fields”](#) on page 247.

◆ To define a field:

1. Place the cursor in the document pane on the character with which you want to start the new field.
2. Click the **Begin Field in This Column** () button. This action is also accessible from the **CustomXMLConversion** menu and the shortcut menu in the Custom XML Conversions Editor.

Stylus Studio displays a thin orange line that identifies the start of the newly defined field.

◆ To remove a field:

The procedure for removing a field is the same as the procedure for defining one – place the cursor on any character adjacent to the field line you want to remove and click the **Begin Field in This Column** () button.

Creating Notes for Fields

Stylus Studio allows you to create notes on individual fields. These notes are for reference purposes only; they are not output in the XML.

◆ **To create notes for a field:**

1. Click the entry field for the **Notes** property.

Tip The **Notes** property is in the **Field Element Name > Source Data Type** tree in the **Properties** window. These properties appear only for rows for which a match pattern exists. See “[Pattern Matching](#)” on page 257 for more information on this topic.

The **Notes** dialog box appears.

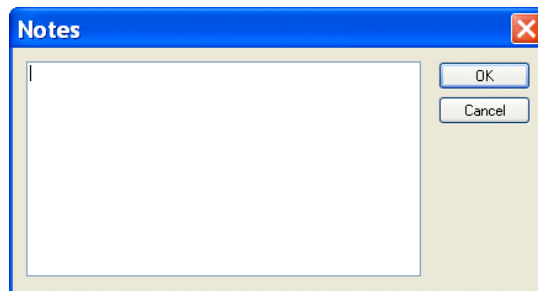


Figure 175. Notes Dialog Box

2. Type the notes you want to associate with the field and click the **OK** button.

Tip If a field has a note defined for it, it is displayed in a tooltip which appears when you hover the mouse pointer over the field in the Custom XML Conversions Editor.

Component and Sub-Component Fields

Some file formats – many EDI variants, for example – allow fields to be subdivided into arrays, sub-fields, or composite fields. Collectively, these fields are referred to as *component fields* in custom XML conversion definitions. Typically, the headers of these files contain information about the character used to specify component fields. Stylus Studio uses this information to set the default value for the **Field Component Separator** property and render XML output accordingly.

Converting Non-XML Files to XML

Consider the following input file, which uses a semi-colon (;) to specify component fields:

```
Make;Model;Year;Color;Seat
BMW;R1150RS;2004;grey,metallic;black,vinyl
MZ;Scorpion;1995;green,clearcoat;black,vinyl
Ducati;ST2;1997;red,clearcoat;black,leather
```

Using the first row to supply the field names and default **Component Element Name** (component), the custom XML conversion definition creates the following XML output (first two elements shown for brevity):

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <row>
    <Make>BMW</Make>
    <Model>R1150RS</Model>
    <Year>2004</Year>
    <Color>
      <component>grey</component>
      <component>metallic</component>
    </Color>
    <Seat>
      <component>black</component>
      <component>vinyl</component>
    </Seat>
  </row>
  <row>
    <Make>MZ</Make>
    <Model>Scorpion</Model>
    <Year>1995</Year>
    <Color>
      <component>green</component>
      <component>clearcoat</component>
    </Color>
    <Seat>
      <component>black</component>
      <component>vinyl</component>
    </Seat>
  </row>
  ...

```

The <component> elements are created as subelements of the <Color> and <Seat> elements.

For line-oriented regions in files containing component fields, you can change the default **Field Component Separator** property, and the **Component Element Name** and **Component Element Name** properties, that is, the name you want to use for the component fields' container elements.

Controlling XML Output

Custom XML conversion definitions provide several ways for you to control the XML output. Most XML output is specified using properties displayed in the **Properties** window. Some XML output, such as the number of regions or the number of fields in a row, are specified using the Custom XML Conversions Editor.

This section describes the properties used to control some of the most common output operations. See [“Custom XML Conversion Definitions Properties Reference”](#) on page 290 for detailed information on all properties.

This section covers the following topics:

- [“Specifying Element Names”](#) on page 255
- [“Specifying Format”](#) on page 256
- [“Omitting Regions and Fields, and Rows”](#) on page 256
- [“Pattern Matching”](#) on page 257
- [“Using Lookup Lists”](#) on page 262
- [“Using Key=Value Characters”](#) on page 265

Specifying Element Names

You can specify names for the following nodes in an XML document output by a custom XML conversion definition:

- Root element – The default for the <root> element is root. You can change the default using the **Root Element Name** property.
- Region element – The default for the <region> element is region. You can change the default using the **Region Element Name** property. Different regions can have different names.
- Namespace – You can specify names for both the namespace prefix and the namespace using the **Namespace** and **Namespace Prefix** properties, respectively. The namespace prefix you specify is added to every element name.
- Row element – The default for the <row> element is row. You can change the default using the **Row Element Name** property. Rows in different regions can have different names.
- Field element – The default for the <field> element is field. You can change the default using the **Field Name** property. Each field can have its own name. If your input file defines subelements, you can use the **Component Element Name** and **Sub-Component Element Name** properties to provide a name for the containing element.

Specifying Format

There are several ways to exercise control over the format of the XML document output by a custom XML conversion definition.

- **Indenting** – By default, Stylus Studio indents the XML generated by a custom XML conversion definition. You can remove indenting by changing the value of the **Indent XML?** property to `False`.
- **Whitespace** – The **Normalize Whitespace** property converts tabs, carriage returns, and line-feeds to spaces. Leading and trailing whitespaces are then removed, and any two or more consecutive whitespaces are collapsed to a single space.
- **XML Schema** – The **XML Schema Document** property allows you to specify the XML Schema you want to associate with the XML output. There are also fields that allow you to specify System and Public DTDs.

Omitting Regions and Fields, and Rows

Stylus Studio allows you to omit specific regions and fields from an input file when it is converted to XML.

- **Omitting regions** – The **Omit from Output** property lets you omit an entire region from XML output. (Regions with a **Region Type** of `No-output` are always omitted from XML output.)
- **Omitting fields** – The **Omit from Output** property lets you omit a field from XML output. You can omit a field
 - Only when it is empty. This is the default.
 - When it is empty or evaluates to a zero value.
 - Always, regardless of its value
 - Never, regardless of its value
- **Comments** – You can filter rows using the **Comment String** property – if the beginning of a row matches the string you enter for this property, Stylus Studio ignores the row when converting the input file to XML. You can also specify patterns that rows must match in order for them to be converted to XML. See [“Pattern Matching”](#) on page 257 for more information.

Pattern Matching

You can use regular expressions to specify match patterns for the rows in the input file. Only those rows in the input file that match the pattern you specify are output to XML when the file is converted. The simplest way to define a match pattern is to use the **Match Pattern** property in the **Row Element Name** section of the **Properties** window.

Example

Consider the following input file:

```
Make, Model, Year, Mileage
BMW, R1150RS, 2004, 14274
Kawasaki, GPz1100, 1996, 60234
Ducati, ST2, 1997, 24000
Moto Guzzi, LeMans, 2001, 12393
BMW, R1150R, 2002, 17439
Ducati, Monster, 2000, 15682
Aprilia, Futura, 2001, 17320
```

If you specify a simple regular expression, say, ^B, for the **Match Pattern** property, Stylus Studio displays the input file in the Custom XML Conversions Editor as shown in [Figure 176](#) – green check marks identify the rows that match the pattern, and red X's identify the rows that do not. (You can also display matching rows in a contrasting color by clicking the **Highlight Matching Rows** button. See [“Document Pane Display Features”](#) on page 231 for more information about this feature.)

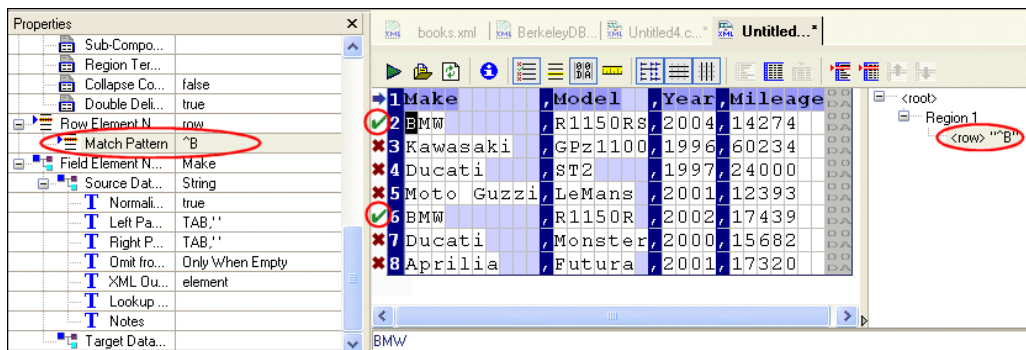


Figure 176. Match Pattern – Definition and Display

Note that the match pattern also appears as a new node in the schema pane. This new node, the only one defined for the custom XML conversion definition at this point, uses the default row element name (`row`) and the value of the expression.

Converting Non-XML Files to XML

Since the match pattern selects only those rows that begin with the letter B, the custom XML conversion definition creates the following XML document when it is run against the input file:

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <row>
    <Make>BMW</Make>
    <Model>R1150RS</Model>
    <Year>2004</Year>
    <Mileage>14274</Mileage>
  </row>
  <row>
    <Make>BMW</Make>
    <Model>R1150R</Model>
    <Year>2002</Year>
    <Mileage>17439</Mileage>
  </row>
</root>
```

See “[Working with Nodes](#)” on page 260 to learn about adding the row element name/match pattern pairs that define them.

Sample Regular Expressions

The following table presents some commonly used regular expressions.

Expression	Matches
<code>^ABC</code>	Match all lines starting with “ABC”
<code>^[Aa][Bb][Cc]</code>	Match all lines starting “ABC”, “abc” or any mix of upper and lowercase (“Abc”, for example)
<code>AAA</code>	Match all lines containing “AAA”
<code>^(DEF GHI)</code>	Match all lines starting with “DEF” or “GHI”
<code>XYZ\$</code>	Match all lines ending with “XYZ”
<code>XYZ\</code>	Match all lines containing “XYZ”

Go to <http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html> to learn about the specific regular expression implementation supported in Stylus Studio. Go

to <http://www.boost.org/libs/regex/doc/syntax.html> for additional examples of regular expression usage.

Specifying Multiple Match Patterns

You can specify multiple match patterns for a single file. If we define a new match pattern, `^K`, this results in a new node (`<row>` “`^K`”) in the schema pane, which now displays both nodes (see [Figure 177](#)). When an input file is converted, Stylus Studio matches the patterns in the order in which the nodes that represent them are defined in the schema. Blank patterns are always matched last.

When you define multiple match patterns, the document pane displays a gray square alongside rows that match a pattern other than the one, if any, associated with the currently selected row. In [Figure 177](#), for example, row 3 is the currently selected row; it matches the pattern `^K` we have defined. Because row 3 is the active row, Stylus Studio displays gray squares in rows 2 and 6 (which match the pattern `B` defined previously).

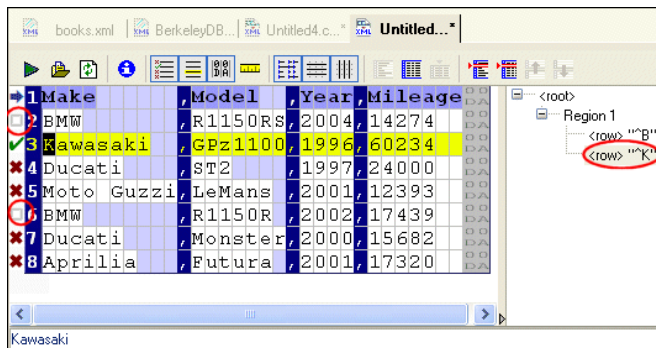


Figure 177. Gray Squares Identify Rows That Match Other Patterns

Working with Nodes

In addition to defining nodes using the **Match Pattern** field of the **Properties** window, you can also use the **Set Node and Match Pattern** dialog box, shown here:

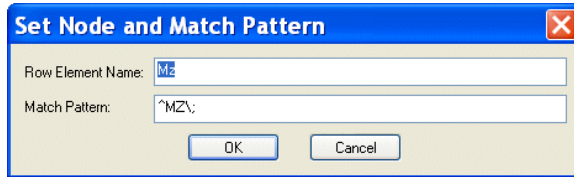


Figure 178. Set Node and Match Pattern Dialog Box

This dialog box allows you to

- Define a new node – even one that does not match a row in the current input file. For example, we could define a match pattern for Triumph motorcycles (<row> “^T”) even though there are no Triumph motorcycles in the input file.
- Clone an existing node – this allows you to copy an existing node and modify its match pattern to create a new node.
- Edit an existing node. (You can also do this in the **Properties** window, of course.)

When you open the dialog box, the **Row Element Name** and **Match Pattern** fields contain default values that reflect the currently selected row in the document pane or node in the schema pane.

Defining a New Node

◆ **To define a new node:**

1. Select a row in the document pane or a node in the schema pane.
2. Select **CustomXMLConversion > Add Node and Pattern** from the Stylus Studio menu.

Alternative: Select **Add Node and Pattern** from the document pane or schema pane shortcut menu.

The **Set Node and Pattern** dialog box appears.

3. Change the default values in the **Row Element Name** and **Match Pattern** fields.
4. Click **OK**.

Cloning a Node

◆ To clone a node:

1. Select the node in the schema pane that you want to clone.
Alternative: Select the row in the document pane that is represented by a row element name/match pattern pair you want to clone.
2. Select **CustomXMLConversion > Clone Node and Pattern** from the Stylus Studio menu.
Alternative: Select **Clone Node and Pattern** from the document pane or schema pane shortcut menu.
The **Set Node and Pattern** dialog box appears.
3. Change the default values in the **Row Element Name** and **Match Pattern** fields as needed.
4. Click **OK**.

Editing a Node

◆ To edit a node:

1. Select the node in the schema pane that you want to edit.
Alternative: Select the row in the document pane that is represented by a row element name/match pattern pair you want to edit.
2. Select **CustomXMLConversion > Edit Node and Pattern** from the Stylus Studio menu.
Alternative: Select **Edit Node and Pattern** from the document pane or schema pane shortcut menu.
Alternative: Double-click the node.
The **Set Node and Pattern** dialog box appears.
3. Change the default values in the **Row Element Name** and **Match Pattern** fields as needed.
4. Click **OK**.

Removing a Node

When you remove a node, you are deleting the row element name/match pattern pair from the custom XML conversion you are defining.

◆ **To remove a node:**

1. Select the node in the schema pane that you want to remove.
Alternative: Select the row in the document pane that is represented by a row element name/match pattern pair you want to remove.
2. Select **CustomXMLConversion > Remove Node and Pattern** from the Stylus Studio menu.
Alternative: Select **Remove Node and Pattern** from the document pane or schema pane shortcut menu.
Alternative: Press the Delete key.
A warning message appears.
3. Click **Yes** to remove the node, otherwise click **No**.

Using Lookup Lists

You can define lookup lists for individual fields. When Stylus Studio converts the input file, it replaces the string in the input file (the lookup) with the value you have defined for it in the **Lookup List** dialog box. [Figure 179](#) shows an example of a lookup list that has been defined for a Status field:

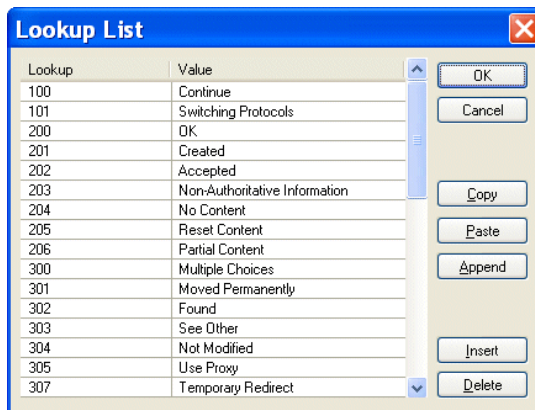


Figure 179. Sample Lookup List

For any Status fields in the input document with a value of, say, 100, Stylus Studio would convert that value to `Continue` in the XML document it outputs; values of 202 would be converted to `Accepted`; and so on.

Input file values that do not match a lookup are emitted in the XML document as-is, allowing exceptional values to be decoded. For example, you might have a temperature lookup list with these values for a <Temperature> field:

```
32 | Freeze  
212 | Boil
```

All other temperatures would be emitted as-is.

Defining Lookup Lists

Lookups are case-sensitive, so, for example, a lookup of `bmw` would not match any of the `Make` fields in the following sample file:

```
Make,Model,Year,Mileage  
BMW,R1150RS,2004,14274  
Kawasaki,GPz1100,1996,60234  
Ducati,ST2,1997,24000  
Moto Guzzi,LeMans,2001,12393  
BMW,R1150R,2002,17439  
Ducati,Monster,2000,15682  
Aprilia,Futura,2001,17320
```

You can define lookup lists only for fields in rows for which a match pattern (even a blank match pattern, as is the default) exists. Finally, you can paste comma- and tab-delimited text directly into the lookup list. This allows you to easily reuse existing lookup tables without having to re-enter text.

◆ To define a lookup list:

1. Select a row for which a match pattern exists.
2. Click the **Lookup List** entry field in the **Properties** window.

The **Lookup List** dialog box appears.

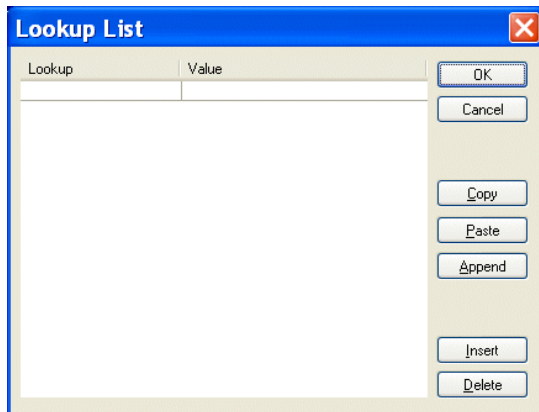


Figure 180. Lookup List Dialog Box

3. Enter lookup/value pairs in the corresponding entry fields.
4. When you are done, click **OK**.

Working with Lookup Lists

The following table summarizes the functions of the **Lookup List** dialog box, which allow you to work with new and existing lookup lists.

Table 22. Lookup List Dialog Box Buttons

<i>Button</i>	<i>Function</i>
OK	Commits the lookup list to the custom XML conversion.
Cancel	Closes the Lookup List dialog box without committing any changes.
Copy	Copies the lookup list.
Paste	Pastes comma- and tab-separated text into the lookup list. Replaces existing content, regardless of which row you have selected
Append	Adds comma- and tab-separated text to the end of the lookup list. Existing lookup list content is preserved.
Insert	Adds a new row to the lookup list.
Delete	Removes the selected row from the lookup list.

Note Copy, Paste, and Append use the system clipboard and insert at the current cursor location. Any blank rows are discarded when you save the lookup list.

Using Key=Value Characters

The **Key=Value Character** Region property allows you to set the separator for key=value pairs as seen in the input file. When Stylus Studio converts an input file to XML, it uses the value on the left side of the key=value character for the element name, and the value on the right for the element value. Consider the following input file:

```
Triumph Inventory,Year and Quantity
Yr2003=24
Yr2004=12
Yr2005=15
```

If you set the **Key=Value Character** property to =, Stylus Studio creates the following XML document when you preview the custom XML conversion:

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <row>
    <field>Triumph Inventory</field>
    <field>Year and Quantity</field>
  </row>
  <row>
    <Yr2003>24</Yr2003>
  </row>
  <row>
    <Yr2004>12</Yr2004>
  </row>
  <row>
    <Yr2005>15</Yr2005>
  </row>
</root>
```

Creating a Custom XML Conversion Definition

This section describes how to create and save a custom XML conversion definition (.conv file) in Stylus Studio.

Tip Stylus Studio comes bundled with built-in DataDirect XML Converters for numerous file formats, including EDI, CSV, dBase, and Windows .ini files. These XML Converters automate the conversion process, and can be quicker and easier to use than building a custom XML conversion definition. See [“DataDirect XML Converters”](#) on page 217 for more information.

Specifying File Settings

When you create a new custom XML conversion definition, Stylus Studio prompts you to specify the file you want to convert. It also allows you to specify the file’s

- Encoding (Windows-1252 or ANSI, for example)
- Layout (line-oriented or fixed-width, for example)

Unless you are certain of the file’s encoding and layout, consider leaving the default settings as they are – Stylus Studio will determine encoding and layout properties when it reads the file.

How to Create a Custom XML Conversion Definition

◆ **To create a custom XML conversion definition (.conv file):**

1. Select **File > New > Custom XML Conversion** from the Stylus Studio menu.
The **New Custom XML Conversion Definition** dialog box appears.

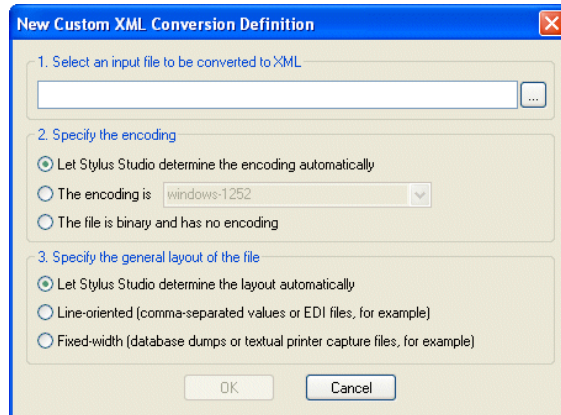




Figure 181. New Custom XML Conversion Definition Dialog Box

2. Click the browse button () and select the file you want to use to configure the custom XML conversion definition settings.

The **Open** dialog box appears. By default, the **Files of Type** field filter is set to display .txt, .edi, .csv, .tab, .log, and .bin files.

Tip The file you select should be representative of other files of this type that you wish to convert to XML using the custom XML conversion definition you are creating.

3. Click the **Open** button.
Stylus Studio displays the file URL in the first field of the **New Custom XML Conversion Definition** dialog box.
4. Optionally, change the default file encoding and layout settings.
5. Click **OK**.
Stylus Studio displays the file you selected in the Custom XML Conversions Editor.
6. Examine the file and its properties as it was read into the Custom XML Conversions Editor by Stylus Studio.

7. Modify the file layout, its regions, and its fields as needed. See “[Working with Regions](#)” on page 239 and “[Working with Fields](#)” on page 247 if you need help with this step.
8. Modify the properties that govern XML output as needed. See “[Controlling XML Output](#)” on page 255 if you need help with this step.
9. Click the **Preview Results** button ().
Stylus Studio displays the **Save As** dialog box.
10. Enter a name for the custom XML conversion definition and click **Save**.
11. If the results are not what you expect, return to [step 7](#) and to [step 8](#). You might also consider revisiting the input file, making fundamental changes there, and then reloading it.

Using Custom XML Conversion Definitions in Stylus Studio

You can use custom XML conversion definitions to open any file as XML anywhere in Stylus Studio. For example, you might want to use a text file (.txt) as the source document for XQuery Mapper. When you open a file using a custom XML conversion definition, the XML Converters engine converts that file to XML on-the-fly, using the settings defined in the custom XML conversion definition you select.

You can also use the DataDirect XML Converters API to invoke a custom XML conversion definition (or an XML Converter). See the documentation for DataDirect XML Converters for more information: <http://www.xmlconverters.com/doc/>.

How to Open a File Using a Custom XML Conversion Definition

You can use the **Open** dialog box to open a file using a custom XML conversion definition in Stylus Studio.

- ◆ **To open a file using a custom XML conversion definition from the Open dialog box:**
 1. Display the **Open** dialog box (select **File > Open** from the Stylus Studio menu, for example).
 2. Navigate to the directory that contains the file you want to open using the custom XML conversion definition and select the file.

3. Check the **Open using XML Converter** check box and click the **Open** button.

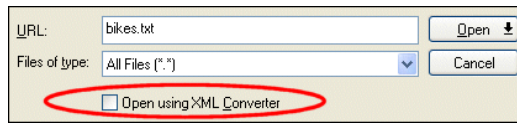


Figure 182. Check Box to Open Files Using XML Converters

Stylus Studio displays the **Select XML Converter** dialog box.

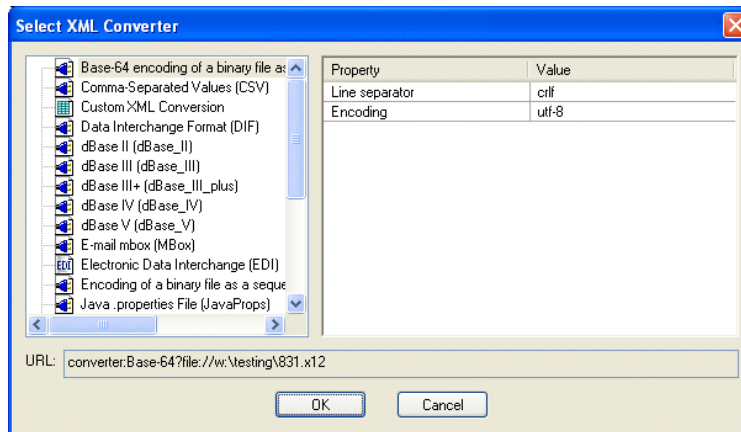


Figure 183. Select XML Converter Dialog Box

Converting Non-XML Files to XML

4. If you are using a custom XML conversion definition (.conv), select **Custom XML Conversions**. Then, use the browse button in the **Value** field to locate the custom XML conversion definition you want to use. Go to [step 5](#).

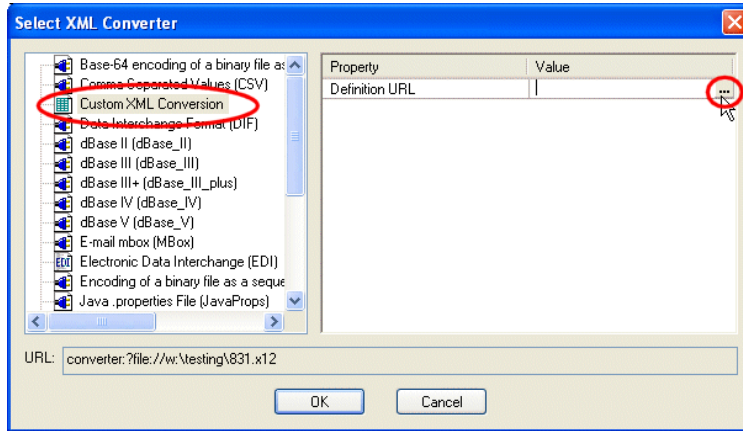


Figure 184. Selecting a Custom XML Conversion Definition

5. Click **OK**.

The file is converted to XML and appears in the editor from which you displayed the **Open** dialog box in [step 1](#).

Tip You can use the same basic procedure, from the **Save As** dialog box, to save a file using a custom XML conversion definition. See [“Saving an XML File in Another Format”](#) on page 222.

Working with EDI Conversions

You can convert EDI to XML (and vice versa) in Stylus Studio using the built-in DataDirect XML Converter for EDI. The EDI XML Converter handles most versions of EDI standard vocabularies (EDIFACT, EANCOM, IATA, and X12) automatically, and it optionally performs validation of content, structure, and code list values.

This section describes more about the level of support for converting EDI to XML (and vice versa) in Stylus Studio.



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the EDI to XML Mapping video](#).

A complete list of the videos demonstrating Stylus Studio's features is here: http://www.stylusstudio.com/xml_videos.html.

This section covers the following topics:

- “Supported EDI Dialects” on page 271
- “Creating Custom EDI Message Types” on page 272
- “Understanding Separator Characters” on page 277
- “Validating XML from/to EDI” on page 283

Supported EDI Dialects

The following EDI dialects are supported for both DataDirect XML Converters and Stylus Studio custom XML conversion definitions.

Table 23. EDI Dialect Support

<i>Dialect</i>	<i>For Detailed Message Type Support See</i>
EDIFACT	http://www.stylusstudio.com/edi/EDIFACT_standard.html
EANCOM	http://www.stylusstudio.com/edi/EANCOM_standard.html
IATA	http://www.stylusstudio.com/edi/IATA_standard.html
X12	http://www.stylusstudio.com/edi/X12_standard.html

Creating Custom EDI Message Types

You can define extensions to standard EDI message types in Stylus Studio. This allows the EDI XML Converter to convert proprietary EDI message types that are based on an extension (or restriction) to a standard EDI message type.

You can create custom EDI message types for

- EDIFACT-style messages (EDIFACT-style includes IATA and EANCOM)
- X12-style messages (X12-style includes ATIS)

The message types you create must be either EDIFACT- or X12-style messages, since the auto-detection feature of the EDI XML Converter depends on the initial segments of those dialects as well as on the general syntax of the document. You cannot, for example, create an HL7 EDI message, since HL7 is structurally different (it allows messages to span documents, it allows segments to be split and continued, it has a sub-field structure, and so on).

Overview

The process for using a custom EDI message type includes the following steps:

1. Create an XML document for the custom EDI message type. The XML document must conform to the `ex.xsd` XML Schema, which is installed with Stylus Studio. (The `ex.xsd` XML Schema is described in detail in the following sections.)
2. Save the resulting XML document.

- Specify the XML file URL in the **Extension map file** property when you specify the properties for the EDI XML Converter in the **Select XML Converter** dialog box.

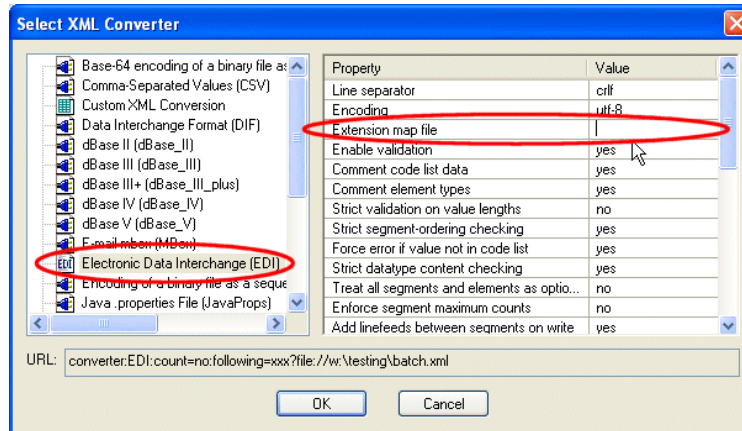


Figure 185. Specifying a Custom EDI Message Type

Specifying the Extension File Location

You can specify the XML document that contains the custom message type definition using

- An absolute URL (c:/mypath/my_extension.xml, for example)
- A relative path (mydir/my_extension.xml, for example)

Note that if you are using a relative path, it must be relative to the same directories in which the XML Converters executables (XMLConverters.jar for the Java version of the XML Converters, XMLConverters.dll for the .NET version).

In environments in which the location of the XMLConverters.* file cannot be determined, you must specify the location:

- For Java – set the system property `com.ddtek.xmlconverter.bindir`, or `com.ddtek.xmlconverter.libdir`
- For .NET – set the registry key `HKLM/Software/DataDirect/XML Converters 3.0/ProductLocation`

The ex.xsd XML Schema

You add custom EDI message types to Stylus Studio by creating XML files that conform to the XML Schema `ex.xsd`. This XML Schema is located in the `\bin\Plugins\Configuration Files` folder where you installed Stylus Studio.

This XML Schema is for a completely described message type, which can include one or more messages, one or more segments, zero or more composites, and one or more elements. The XML Schema assumes that all pieces needed are described within the XML.

The Root Element

The root element of the custom EDI message type XML document is `<edi>`. Within the root element, you can define each of the following components: `<message>`, `<segment>`, `<composite>`, and `<element>`. When one component references another, it will always be in the form of a `<use-...>` element. For example, a `<message>` will have one or more `<use-segment>` elements within it, which will refer to the `<segment>`s defined elsewhere in the file.

The Message Element

Each EDI message is composed of one or more `<message>` elements (or messages). Within each message there are groups of segments, which can be nested, so each group can contain other groups, segments, or a mix. Within each segment there can be a mix of composite fields and elements, and composite fields are also made up of elements.

The `<message>` element will have a mandatory attribute, `@name`, which is the name of the EDI message. This must match what is found in the file in a UNH/UIH segment for EDIFACT, or in an ST segment for X12. Something like "ORDERS" or "811", for example.

The `<message>` element should also have a `@title` attribute, which sets the human-readable title. It can also have a child `<description>` element, which can contain an arbitrarily long description.

Next should be one or more `<group>` or `<use-segment>` elements. They can be mixed, although order does matter. The `<use-segment>` elements have three attributes:

- `@name`, which contains the name of a `<segment>` elsewhere in the file (or elsewhere in the internal standards if the keyref-less version of the schema is being used to augment an existing EDI definition)

- @mandatory, whose "true" or "false" value indicates whether the segment is mandatory
- @count, which indicates the maximum number of times the segment may appear. Note that the count must be greater than zero, and @mandatory="true" means that at least one must appear – if @count="99" and @mandatory="true" it does not mean that 99 are required, however.

The Group Element

The <group> element can contain one or more <group> and/or <use-segment> elements. The <group> element allows also an optional <description> element for documentation purposes.

Group elements also have two attributes, @count and @mandatory, which behave as described in [“The Message Element”](#) on page 274.

The Segment Element

The <segment> element, like messages, has @name and @title attributes. The name should be the name of the segment as it will appear in the EDI document, such as "NAD" or "F6X" or "ID1", for example. It also supports the optional <description> element.

Each <segment> contains one or more elements – either simple elements referred to by the <use-element> construct, or component elements referred to by the <use-composite> construct. The order is very significant. Both <use-element> and <use-composite> take two attributes, @name and @mandatory:

- @name points to an <element> (in the former case) or <composite> (in the latter) within the file (or elsewhere in the internal standards if the keyref-less version of the schema is being used to augment an existing EDI definition)
- @mandatory is whether the element or component must be present. Often in EDI messages, an element or composite can be left out, either by putting two delimiters together with no content or by eliminating unneeded fields from the end of a segment. This attribute indicates whether that practice is allowed for that particular element/composite.

Composite Fields

A `<composite>` element is like a `<segment>`, except that it cannot contain `<use-composite>` elements. It can only contain `<use-element>` elements. It is used to build up more complicated structures of related information. One use is to hook together a date and a time field, or a date and a date-format and a purpose-of-date.

Composite names usually look like C???, E??? or S???, with the ??? being a three-digit number. While this is true for both X12 and EDIFACT, it is neither a rule nor a requirement.

Elements

An `<element>` describes the most basic component of an EDI message.

Element names are typically numbers, although X12 has a number of elements in which the first character is the letter 'I' followed by a two-digit number. The name is in the mandatory `@name` attribute. The `@title` attribute is a human-readable description of the element's purpose.

The mandatory `@type` attribute describes the datatype. EDIFACT and X12 both use:

- 'a' (alphabetical)
- 'an' (alphanumeric)
- 'n' (numerical)

X12 expands on the list of types with these additional types:

- 'id' (identifier - this must be contained within a code list)
- 'b' (binary - raw bits)
- 'dt' (date, in CCYYMMDD format)
- 'n0' (same as 'n')
- 'n1' through 'n9' (these are numeric, but scaled - the digit tells how many places to the left to move the decimal – so that a "123" stored as 'n2' would be exported as 1.23 when going from EDI to XML, and '1.23' in XML would be rendered in the EDI as '123')

The mandatory `@maxLength` attribute indicates how many characters at most can be in the value. The optional `@minLength` sets the minimum length. The default for the latter is "0".

Optionally, each element can have its values further constrained by an enumerated list. That list is denoted by a series of `<item>` elements within the `<element>`.

Each <item> has a @key attribute which holds the value that would appear in the EDI, and the text content of the <item> element is the description of that value.

Understanding Separator Characters

An EDI file consists of tokens terminated by separators. Special characters are used to define separators and other symbols, like the decimal character. You can specify the characters used for the following EDI file items:

- Segment terminator
- Element separator
- Component value separator
- Subcomponent (tertiary) separator
- Segment name/segment content separator
- Release (escape) character
- Decimal character
- Repeat symbol

Values for some of these items are set explicitly in the EDI data stream. For example, in X12, the component separator is the value of the ISA16 field (the "I15: Component Element Separator"). In EDIFACT, separator values can be set using the contents of the UNA segment.

How Stylus Studio Sets Separator Characters

When writing XML to EDI, Stylus Studio overwrites separator values using a set of processing instructions (PIs) in the XML. As long as the character specified in the PI has not been emitted to the output stream, it can be changed. For example, the segment terminator can be changed using the PI only *before* the first segment is written to EDI.

Note In X12, since the I15 element contains the component separator, it cannot be set by a PI as one value would conflict with the other.

Setting Processing Instruction

Stylus Studio uses the following naming convention and default values for XML processing instructions used to set values for separator characters:

Table 24. Processing Instructions for Separator Characters

<i>Name</i>	<i>Default Value</i>
edi_segment	'
edi_release	?
edi_element	+
edi_component	:
edi_repeat	~
edi_decimal	, Note that the decimal character is read from the underlying platform default. If no default is set, the EDIFACT standard specifies a comma.
edi_tertiary	&
edi_follow	=

If a PI is used to set the value for one separator (say, edi_A) to a default value being used by another separator (edi_B), a new value will be chosen for the edi_B separator unless one has already been specified.

URI switches are available for each of these processing instructions. See the DataDirect XML Converters documentation at <http://www.xmlconverters.com/doc/>. for more information.

Syntax

The syntax of the PI is just "<?" followed by the name, followed by a space, and then the new special character before a closing ">" tag. Consider a few examples:

```
<?edi_segment \n?>
<?edi_release \\?>
<?edi_element $?>
<?edi_component \x0001?>
<?edi_decimal .?>
```

Here is another example, showing PIs used in context. In this example, an X12 document needed to be written to meet the following requirements:

- The segment terminator needs to be a carriage return
- The element separator needs to be an asterisk
- The component separator needs to be a greater-than symbol.

The two PIs and the Component Element Separator are shown in bold for clarity:

```
<?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>
  ...

```

If the segment terminator (edi_segment) is set to \r or \n, consider changing the default value for the **Add linefeeds between segments on write** property. Otherwise, the EDI output will have double endings. See the DataDirect XML Converters documentation for complete properties reference information: <http://www.xmlconverters.com/doc/>.

Ways to Specify Control Characters

In addition to literal characters, you can also specify control characters using the values in the following table. The values you set for these properties apply only when converting XML to EDI. Not all EDI dialects use all special characters. Finally, you must use unique values for each property you choose to set.

Table 25. Special Characters for Separators

<i>Character</i>	<i>Decimal</i>	<i>Hex</i>	<i>Other</i>
NUL	\d0	\u0	
SOH	\d1	\u1	
STX	\d2	\u2	
ETX	\d3	\u3	
EOT	\d4	\u4	
ENQ	\d5	\u5	
ACK	\d6	\u6	
BEL	\d7	\u7	
BELL	\d7	\u7	
BS	\d8	\u8	
HT	\d9	\u9	\t
TAB	\d9	\u9	\t
LF	\d10	\uA	\n
VT	\d11	\uB	
FF	\d12	\uC	\f
CR	\d13	\uD	\r
SO	\d14	\uE	
SI	\d15	\uF	
DLE	\d16	\u10	
DC1 (XON)	\d17	\u11	
DC2	\d18	\u12	
DC3 (XOFF)	\d19	\u13	
DC4	\d\0	\u14	

Table 25. Special Characters for Separators

<i>Character</i>	<i>Decimal</i>	<i>Hex</i>	<i>Other</i>
NAK	\d21	\u15	
SYN	\d22	\u16	
ETB	\d23	\u17	
CAN	\d24	\u17	
EM	\d25	\u19	
SUB	\d26	\u1a	
ESC	\d27	\u1b	
FS	\d28	\u1c	
GS	\d29	\u1d	
RS	\d30	\u1e	
US	\d31	\u1f	
DEL	\d127	\u7F	
BPH	\d130	\u82	
NBH	\d131	\u83	
IND	\d132	\u84	
NEL	\d133	\u85	
SSA	\d134	\u86	
ESA	\d135	\u87	
HTS	\d136	\u88	
HTJ	\d137	\u89	
VTS	\d138	\u8A	
PLD	\d139	\u8B	
PLU	\d140	\u8C	

Table 25. Special Characters for Separators

<i>Character</i>	<i>Decimal</i>	<i>Hex</i>	<i>Other</i>
RI	\d141	\u8D	
SS2	\d142	\u8E	
SS3	\d143	\u8F	
DCS	\d144	\u90	
PU1	\d145	\u91	
PU2	\d146	\u92	
STS	\d147	\u93	
CCH	\d148	\u94	
MW	\d149	\u95	
SPA	\d150	\u96	
EPA	\d151	\u97	
SOS	\d152	\u98	
SCI	\d154	\u9A	
CSI	\d155	\u9B	
ST	\d156	\u9C	
OSC	\d157	\u9D	
PM	\d158	\u9E	
APC	\d159	\u9F	
NBS (NBSP)	\d160	\uA0	
SHY	\d173	\uAD	

Invalid Separator Characters

You cannot use any of the following characters as EDI separators:

- Spaces
- Alphabetical characters
- Numeric characters

Validating XML from/to EDI

Stylus Studio provides several document wizards that convert various EDI dialects and message types to XML Schema. Such an XML Schema can be useful if you are converting XML to EDI because it allows you to ensure that the EDI created by the Electronic Data Interchange (EDI) XML Converter conforms to a particular EDI message type standard. There are document wizards for all EDI dialects supported in Stylus Studio: EDIFACT, EANCOM, IATA, and X12.

You can also use the XML Schema created by the various EDI to XML Schema document wizards to validate data after you have converted EDI to XML, when URL-based error checking is disabled, to determine how well the incoming stream conforms to the EDIFACT standards.

See [Creating XML Schema from EDI](#) on page 519 for more information on using these document wizards.

The Converter URL Scheme

You can use the converter: URL scheme to reach a variety of data sources using DataDirect XML Converters and custom XML conversion definitions.

Where You Use Converter URLs

You can use converter URLs as the argument in XQuery and XSLT 2.0 `doc()` functions, and in XSLT 1.0 `document()` functions. The following example of the `document()` function, which invokes the CSV XML Converter to convert the file `one.csv` to XML:

```
document('converter:///CSV:sep=,:first=yes?' + exampleDir + @"one.csv')
```

Specifying a Converter URL

To specify a converter: URL, you need to identify

- The XML Converter (EDI, CSV, dBase, and so on) or custom XML conversion definition (.conv file) you want to use.
- Options for the XML Converter (separator and escape characters, for example). Options for custom XML conversion are part of the custom XML conversion definition and are not specified.
- The file to be converted.

Example – Converter URL with a DataDirect XML Converter

A converter: URL that invokes the DataDirect XML Converter for comma-separated values (CSV) to convert the `three.txt` file in the `Stylus Studio\examples` directory to XML might look like this:

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

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

Table 26. URL Scheme Example

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

Properties that use default values (a comma is the default separator character for the CSV XML Converter, for example) do not have to be specified in a converter URL.

Example – Converter URL with a Custom XML Conversion Definition

A converter URL that references a custom XML conversion definition might look like this:

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

This converter uses `myConverter.conv` to convert the file `inventory.txt` to some format (specified in the custom XML conversion definition when it was built using the Custom XML Conversion Definition Editor in Stylus Studio). Note that individual configuration properties are not specified – they are part of the definition described in the `.conv` file.

Converter URL Syntax

The converter URL syntax is the same whether you are using them for DataDirect XML Converters or custom XML conversion definitions.

```
converter:name:[property_name=value: | property_name=value: | ...]?file:file URL
```

Where:

- *name* is the name of the DataDirect XML Converter or the `.conv` file name of a custom XML conversion definition.

XML Converter names are displayed in Stylus Studio; see “[Where Converter URLs are Displayed in Stylus Studio](#)” on page 287. They are also described in the DataDirect XML Converters documentation; see <http://www.xmlconverters.com/doc/>.

- *property_name* is the name of a DataDirect XML Converter property. You need to specify a property and its value only if you want to configure the converter to use something other than the default value. You do not specify properties for custom XML conversion definitions.

XML Converter properties are displayed in Stylus Studio; see “[Where Converter URLs are Displayed in Stylus Studio](#)” on page 287. They are also described in the DataDirect XML Converters documentation; see <http://www.xmlconverters.com/doc/>.

- *file URL* is the URL of the file you want to convert.

XML Converter Properties

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

In addition, property names in converter URLs appear in an abbreviated format – the *Line separator* property is called *newline* in a converter URL.

See the DataDirect XML Converters documentation for complete properties reference information: <http://www.xmlconverters.com/doc/>.

Where Converter URLs are Displayed in Stylus Studio

Converter URLs are displayed in the following places in Stylus Studio:

- The **URL** field of the **Select XML Converter** dialog box displays the converter URL for the XML Converter or custom XML conversion definition you have selected.

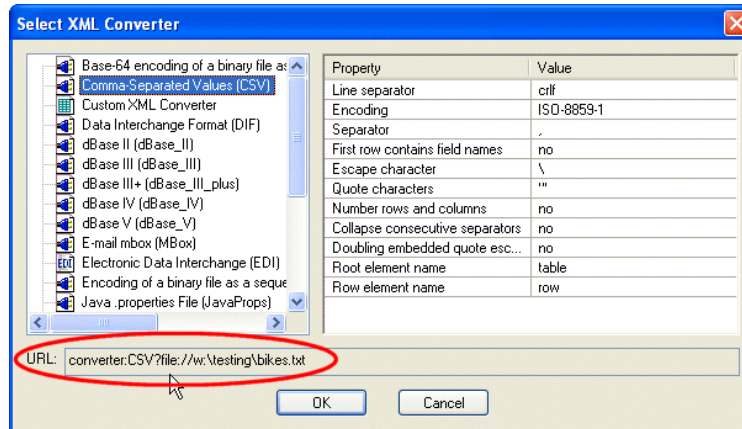


Figure 186. Converter URL Displayed in the Select XML Converter Dialog Box

Note that XML Converter name and properties used in the converter URL vary from the full names displayed in the **Select XML Converter** dialog box. In addition, note that only properties whose default value you change in the **Value** field are displayed in the **URL** field.

- The **Project** window (select **Show Full URL** from the **Project** window shortcut menu) displays the converter URL used to create an XML document:

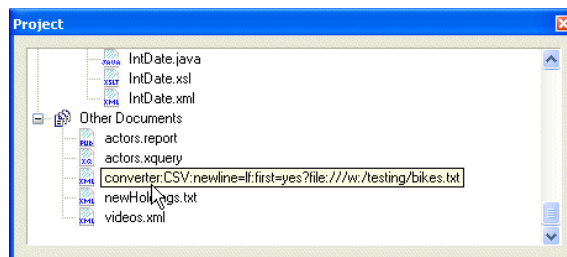


Figure 187. Converter URL Displayed in Project Window

You can use either source for the converter URL strings you want to use in your XQuery and XSLT code. See [“Using Stylus Studio to Build a Converter URL”](#) on page 288 for more information.

Tip Converter URLs in the **Select XML Converter** dialog box are already escaped and can be used as-is. Converter URLs taken from the **Project** window must be escaped manually when you paste them into your XSLT or XQuery code.

Using Stylus Studio to Build a Converter URL

If you have Stylus Studio 2007 XML Enterprise Suite, you can use Stylus Studio to construct converter URLs. Converter URLs can be complex – property names and their values vary from one XML Converter to another, for example – so using Stylus Studio to construct them can reduce errors in your applications.

Using the URL in the Select XML Converter Dialog Box

- ◆ **To construct a converter URL using the URL in the Select XML Converter dialog box:**
 1. Use the XML Converter or custom XML conversion definition to open a file as an XML document in Stylus Studio. See [“Using an XML Converter to Open a Non-XML File as XML”](#) on page 221 or [“How to Open a File Using a Custom XML Conversion Definition”](#) on page 268 if you need help with this step.
 2. Before clicking **OK** to complete the conversion, copy the converter URL in the **URL** field of the **Select XML Converter** dialog box (see [Figure 186](#)).
 3. Click **OK** to complete the conversion. (You can click **Cancel** if you are performing this procedure just to obtain the converter URL.)
 4. Paste the converter URL in your XSLT or XQuery code as needed.

Using the URL in the Properties Window

◆ To construct a converter URL using the URL in the Properties window:

1. Use the XML Converter or custom XML conversion definition to open a file as an XML document in Stylus Studio. See “Using an XML Converter to Open a Non-XML File as XML” on page 221 or “How to Open a File Using a Custom XML Conversion Definition” on page 268 if you need help with this step.

Tip New documents are placed in the **Other Documents** folder in the **Project** window by default.

2. Open a new document in any Stylus Studio text editor (for example, **File > New > XML Document**).

The purpose of this step is to provide an editor into which you can drag-and-drop the the document you created in [step 1](#) in order to display the associated converter URL.

3. Drag the document you created in [step 1](#) from the **Project** window and drop it into the text editor you opened in [step 2](#).

The complete URL appears in the text editor.

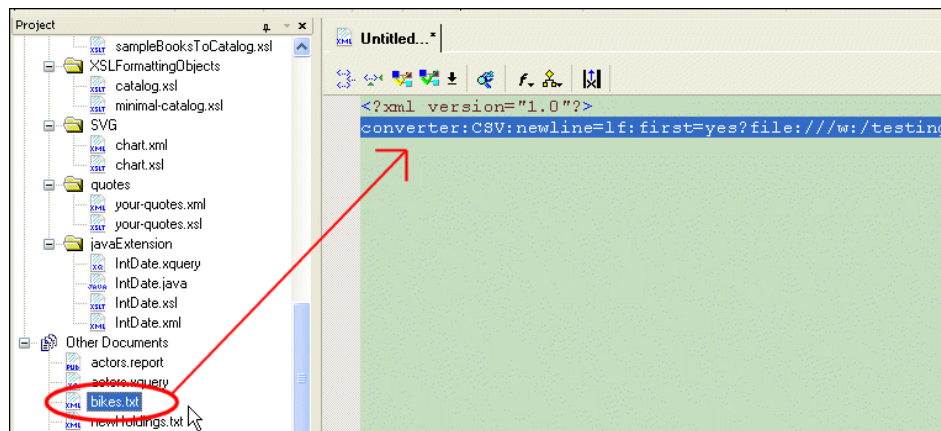


Figure 188. Copying a URL to a Text Editor

4. Copy the complete converter URL.
5. Paste the converter URL in your XSLT or XQuery code as needed.

Note: Escape characters as required for strings in Java programs. For example, `escape=:quotes=''` becomes `escape=\\:quotes='\"` (the single quote does not need to be escaped).

Custom XML Conversion Definitions Properties Reference

This section provides reference information for properties displayed in the **Properties** window of the Custom XML Conversions Editor. It covers the following topics:

- [“Input File Properties”](#) on page 290
- [“XML Output URL Properties”](#) on page 291
- [“Region Type Properties”](#) on page 293
- [“Row Element Name Properties”](#) on page 296
- [“Field Element Name Properties”](#) on page 297
- [“Data Type Properties \(by data type\)”](#) on page 299
- [“Specifying Control Characters”](#) on page 320

Input File Properties

Table 27. Input File Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Input File	The URL of the file you are using as the input file.	Yes. You can select a new input file from this field.	No
Input Encoding	Encoding detected by Stylus Studio.	Yes	No
Physical Size	The size of the input file in bytes.	No	No
Portion Loaded	The size of the input file in bytes loaded into Stylus Studio.	No	No

Table 27. Input File Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Characters Loaded	The number of characters loaded. For especially large files, Stylus Studio does not read the entire file as only a sample is required to define a custom XML conversion definition. The finished custom XML conversion definition, however, reads any file you open it with in its entirety.	No	No
Character Unicode Value	The Unicode value of the character under the cursor.	No	No

XML Output URL Properties

Table 28. XML Output File Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
XML Output URL	The URL to which you want the XML output when the custom XML conversion definition is run. Optional. If no value is specified, stdout is used.	Yes	Yes
Output Encoding	The Encoding you want to use for the XML output by the custom XML conversion definition. The default is RAW.	Yes	Yes
Root Element Name	The name you want to assign to the root element in the XML output. Optional.	Yes	Yes
Namespace Prefix	The string you want to use for the namespace prefix in the XML output. Optional	Yes	Yes

Table 28. XML Output File Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Namespace	The namespace you want to use for the XML document output by the custom XML conversion definition.	Yes	Yes
DOCTYPE System ID	The URL of the System DTD you want to associate with the XML document output by the custom XML conversion definition.	Yes	Yes
DOCTYPE Public ID	The Public ID of the DTD you want to associate with the XML document output by the custom XML conversion definition.	Yes	Yes
XML Schema Document	The URL of the XML Schema document you want to associate with the XML document output by the custom XML conversion definition.	Yes	Yes
XML Schema Namespace	The namespace you want to use with the XML Schema document.	Yes	Yes
Indent XML?	Whether or not you want to indent the XML output.	Yes	Yes

Region Type Properties

Note Each region has its own field and row properties. In addition, some properties apply only to line-oriented regions. These properties are marked with an asterisk.

Table 29. Region Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Region Type	The type of the region that currently has focus.	Yes. You can change the region type from this field.	No
Region Element Name	The name you want to assign to this region in the XML output. Optional.	Yes	Yes
Element Name Source	Whether the source for the element names in this region is user-supplied, is taken from the first row in the region, or is based on the WS-EDI standard.	Yes	Yes
Rows to Skip	The number of rows, starting at the beginning of the region, you want to omit from output.	Yes	Yes
Omit from Output	Whether or not you want to omit the entire region from output.	Yes	Yes
Size	The region's size in characters.	No	No
Portion of File	The starting and ending offsets of the current region.	No	No
Row Count	The number of rows in the current region.	No	No
Row Length ⁺	The number of characters in the current row.	No	No
Line Terminator*	The type of line terminator character detected by Stylus Studio.	Yes	No

Table 29. Region Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Separator*	The type of field separator character detected by Stylus Studio.	Yes	No
Delimiter Pairs*	Sets of delimiting characters detected by Stylus Studio.	Yes	No
Comment String	String used by the custom XML conversion definition – if the beginning of a row matches the string in this field, the converter interprets the row as a comment and does not output it in the XML.	Yes	Yes
Escape Characters*	Allows you to distinguish escape characters from separators – if a character in the input file is preceded by an Escape Character, that character is not treated as delimiting character, separator, or subsequent escape character.	Yes	Yes
Toss Characters*	Characters outside delimiting characters that should be ignored.	Yes	Yes
Key=Value Character	Allows you to set the separator for key=value pairs as seen in the input file.	Yes	Yes
Component Separator*	The type of character used to separate sub-fields detected by Stylus Studio. If this character appears in a string, the string is split into sub-fields when the input file is converted to XML.	Yes	Yes
Sub-Component Separator*	The type of character used to separate sub-fields detected by Stylus Studio. If this character appears in a string, the string is split into sub-fields when the input file is converted to XML.	Yes	Yes

Table 29. Region Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Region Terminator	The type of region terminator character detected by Stylus Studio. The region will be processed until this string is encountered; after that point, all remaining data in the region is skipped. The next region, if present, will be handled immediately.	Yes	No
Collapse Consecutive Field Separators*	Whether or not multiple consecutive field separators should be treated as one. For example, if this property is set to Yes , X, ,Y, ,Z is treated as X,Y,Z. This property is most useful when spaces are used as delimiting characters.	Yes	Yes
Double Delimiter to Escape	Whether or not to treat a pair of delimiting characters within a delimited string as escaped characters. For example, if this property is set to Yes , "abc""xyz" is treated as abc"xyz.	Yes	Yes

* This property is for line-oriented regions only.

+ This property is for fixed-width regions only.

Row Element Name Properties

Table 30. Row Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Row Element Name	The name you want to assign to all rows in the region. The default value is row.	Yes	Yes
Match Pattern	A regular expression you can use to filter rows in a region. Only rows that match the pattern you specify are output to XML.	Yes	Yes
Current Row Length	The length of the current row.	No	No
Fields in Current Row	The number of fields in the current row.	No	No
Max Fields in Row	The number of fields in the row that contains the largest number of fields.	No	No

Field Element Name Properties

Note Each region has its own field and row properties. In addition, some properties apply only to line-oriented regions. These properties are marked with an asterisk.

Table 31. Field Properties

Property	Description	Editable	Affects XML
Field Element Name	The name you want to use for field elements in the XML output. The default value is <code>field</code> .	Yes	Yes
Component Element Name	The name you want to use for sub-fields detected by Stylus Studio (based on the Component Separator character) in the XML output. The default value is <code>component</code> .	Yes	Yes
Sub-Component Element Name	The name you want to use for sub-fields detected by Stylus Studio (based on the Sub-Component Separator character) in the XML output. The default value is <code>subcomponent</code> .	Yes	Yes
Source Data Type	The data type of the current field. Stylus Studio provides support for the following data types: String, Boolean, number, date, time, byte, short, integer, long, float, double. Note that some properties are type-specific. See “Data Type Properties (by data type)” on page 299 for information on properties that are associated with specific data types.	No	Yes
Target Data Type	<i>Not currently used.</i>		

Table 31. Field Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Number	The number of the field in which the cursor is located. Starting with 1 from the left-most field.	No	No
Cursor Position	The offset of the cursor's current location from the start of the current field.	No	No
Offset	The offset of the start of the current field. Measured from the start of the row.	No	No
Length	The length of the current field in characters.	No	No
Max Field Length *	The length of the longest of all the instances of this field.	No	No
Value	The value of the current field. This value also appears in the status bar.	No	No

* This property is for line-oriented regions only.

Data Type Properties (by data type)

This section describes the properties that are specific to a given data type.

Common Properties

All data types support these properties:

Table 32. Common Properties

<i>Property</i>	<i>Description</i>	<i>Affects XML</i>
Lookup List	This property lets you build a substitution table for output. It includes a two-column table. At runtime, any item which evaluates to a value in the left column is replaced by the corresponding value in the right column in the output stream. A blank value on the left can be used to set a default value for the field. Note that this is applied before the test to determine if the data should be qualify for Omit from Output.	Yes
Notes	Allows you to add internal comments for a field. It is not used by custom XML conversion definitions.	No
Omit from Output	Allows a field to be omitted from XML output based on its presence or value. Valid values include: <ul style="list-style-type: none"> ● Only When Empty. This is the default. ● When Empty or Zero. Does not emit an element containing the value if it evaluates numerically to zero. ● Always. Always hides the field from the output. ● Never. Always includes the field in the output 	Yes
XML Output Form	Determines whether the value is emitted into the output stream as an element or as an attribute. The default is element.	Yes

BCD Datatype Properties

Binary Coded Decimal (BCD) is a set of ways to pack one or two decimal digits into each byte. These are various related types of machine-specific encodings for numbers. The Comp3 and Zoned types are different implementations of this idea.

Table 33. BCD Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Architecture	Select the architecture that matches the machine type that the data originated on, or was designed for, from this list: <ul style="list-style-type: none"> ● NBCD Signed ● NBCD Unsigned ● Excess-3 ● BCD 2421 ● BCD 84-2-1 ● IBM 1401 Signed ● IBM 1401 Unsigned 	Yes	Yes
Packed	The name you want to use for sub-fields detected by Stylus Studio (based on the Component Separator character) in the XML output. The default value is component.	Yes	Yes
Scaling Factor (10 ⁿ)	The name you want to use for sub-fields detected by Stylus Studio (based on the Sub-Component Separator character) in the XML output. The default value is subcomponent.	Yes	Yes

Binary Datatype Properties

Binary data in raw form.

Table 34. Binary Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Rendering	<p>How the data is written to XML:</p> <ul style="list-style-type: none"> ● Base-64 Encoding – writes the data in a form compatible with the W3C XML Schema base64Binary type (default) ● Hexadecimal Encoding – writes the data in a form compatible with the W3C XML Schema hexBinary type ● – Octal Encoding – writes the data as a series of octal triplets ● # Literal Encoding – copies the data as-is to the output stream, which may not render as valid XML 	Yes	Yes

Boolean Datatype Properties

True/false values, with support for three-valued-logic (true/false/unknown or null). The following steps are taken to determine the value of a boolean field:

1. If the field contains all binary zeros, it is false.
2. If the field contains all binary ones (that is, all 0xFF values), it is true.
3. If the field is one byte long and contains a 0x01 value, it is true.
4. If the first or last byte in the field contains a 0x01 and all of the other bytes are 0x00, it is true.
5. If the contents of the field match any of the items in the True Value Match List or False Value Match List, then the value is true or false respectively.

6. If none of these rules apply, the value is considered unknown.

Table 35. Boolean Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
True Value Match List	<p>A semicolon (";") -separated list of values. If the input value matches any of them in step 5, it is considered true and the value from the True Output As property is emitted. The default list is:</p> <ul style="list-style-type: none"> ● y ● t ● yes ● true ● .y. ● .t. <p>The comparisons are not case-sensitive.</p>	Yes	Yes
False Value Match List	<p>A semicolon (";") -separated list of values. If the input value matches any of them in step 5, it is considered false and the value from the False Output As property is emitted. The default list is:</p> <ul style="list-style-type: none"> ● n ● f ● no ● false ● .n. ● .f. <p>The comparisons are not case-sensitive.</p>	Yes	Yes
True Output As	If the value is determined to be true then this value is output. The default is "Yes".	Yes	Yes
False Output As	If the value is determined to be false then this value is output. The default is "No".	Yes	Yes

Table 35. Boolean Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Unknown Output As	If the value cannot be determined as true or false, then this value is output. The default is "?".	Yes	Yes
Left Padding, Right Padding	<p>These two properties control what sort of padding is to be found on either side of the field and then removed. Any characters listed here will be removed. Characters can be specified either by their Unicode values, their Unicode names, or single-quoted.</p> <p>The default for both is spaces (0x20) and tabs (0x09).</p> <p>If there are field delimiters, these are the characters that are trimmed from the contents within the delimiters. The region-level Characters to Toss setting determines which characters are trimmed outside of the delimiters. If there are no delimiters, then the Characters to Toss setting is applied first, and then Left Padding and Right Padding are applied.</p>	Yes	Yes

Byte Datatype Properties

A single byte with a range of 0 to 255 for unsigned or -128 to 127 for signed.

Table 36. Byte Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Signed	<ul style="list-style-type: none"> ● true (default) ● false – use this to make the value unsigned 	Yes	Yes
Scaling Factor (10^n)	<p>This is a value from -16 (shifts the decimal 16 places to the right, making the number smaller by n orders of magnitude) to 0 (keeps the number exactly as it appears in the input) to 16 (shifts the decimal 16 places to the left, making the number larger by n orders of magnitude). To enter a scaling factors from 10^{-16} to 10^{16}, just enter the exponent value.</p> <p>Example: Entering an exponent of 3 will cause any numbers to be multiplied by 1000 (10^3) before being output to XML.</p>	Yes	Yes

Comp3 Datatype Properties

Also known as "Computational-3," "Packed" or "Packed Decimal." A COBOL storage format similar to Zoned below but differing in internal structure. It is also related to the BCD types.

Table 37. Comp3 Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Scaling Factor (10^n)	This is a value from -16 (shifts the decimal 16 places to the right, making the number smaller by n orders of magnitude) to 0 (keeps the number exactly as it appears in the input) to 16 (shifts the decimal 16 places to the left, making the number larger by n orders of magnitude). To enter a scaling factors from 10 ⁻¹⁶ to 10 ¹⁶ , just enter the exponent value. Example: Entering an exponent of 3 will cause any numbers to be multiplied by 1000 (10 ³) before being output to XML.	Yes	Yes

Date Datatype Properties

A date, stored either as a string or in binary form.

The parsing rules are:

1. If the length is 3, then the day, month and year are read as binary values in the Date Format order. The Window for Two-digit Years value is used to determine the century. If the format is YJ, then the first byte is the year, and the second two are the day number in the year, stored as LSB, MSB.
2. If the length is 6, then the same is used, except each day/month/year is taken as two digits.
3. If the length is 8 and there are no separators, then the day and month each get two digits and the year gets four.
4. Otherwise, it is parsed using the YMD Separator characters.

5. For months, the names or abbreviations in English, German, Spanish, French, Portuguese and Italian are recognized.

Table 38. Date Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Date Format	<p>The default is read from the local system settings at the time the custom XML converter is defined.</p> <ul style="list-style-type: none">● DMY (day-month-year)● MDY (month-day-year)● YMD (year-month-day)● YJ (year-daynumber a.k.a. "Julian" format)	Yes	Yes
YMD Separator	<p>A hint to the parser as to the most likely separator between the date components that will be seen.</p> <p>The default value comes from the local system configuration, and also includes the period ("."), comma (","), hyphen ("-"), slash ("/") and space.</p>	Yes	Yes

Table 38. Date Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Window for Two-digit Years	If the year is only given as two digit, this is the cut-off date for determining the century. The default is 1950.	Yes	Yes
Left Padding, Right Padding	<p>These two properties control what sort of padding is to be found on either side of the field and then removed. Any characters listed here will be removed. Characters can be specified either by their Unicode values, their Unicode names, or single-quoted.</p> <p>The default for both is spaces (0x20) and tabs (0x09).</p> <p>If there are field delimiters, these are the characters that are trimmed from the contents within the delimiters. The region-level Characters to Toss setting determines which characters are trimmed outside of the delimiters. If there are no delimiters, then the Characters to Toss setting is applied first, and then Left Padding and Right Padding are applied.</p>	Yes	Yes

DateTime Datatype Properties

A date plus a time.

This is parsed as a date followed by the Date-Time Separator followed by a time. It combines the properties of both, and includes that one additional property.

Table 39. DateTime Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Date-Time Separator	This is the character that separates the date string from the time string. The default values are 'T' and ' '.	Yes	Yes

Decimal Datatype Properties

Reads 16 bytes and interprets it as a .net System.Decimal. Numbers as large as 1028 (positive or negative) and with as many as 28 significant digits can be stored as a decimal type without loss of precision.

Double Datatype Properties

The standard IEEE 8-byte floating point format.

Table 40. Double Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Scaling Factor (10^n)	<p>This is a value from -16 (shifts the decimal 16 places to the right, making the number smaller by n orders of magnitude) to 0 (keeps the number exactly as it appears in the input) to 16 (shifts the decimal 16 places to the left, making the number larger by n orders of magnitude). To enter a scaling factors from 10^{-16} to 10^{16}, just enter the exponent value.</p> <p>Example: Entering an exponent of 3 will cause any numbers to be multiplied by 1000 (10^3) before being output to XML.</p>	Yes	Yes
Endian	<p>When binary numbers are stored, they can be stored with the smallest component first (little-endian) or last (big-endian).</p> <ul style="list-style-type: none"> ● Little – the standard for Intel x86-based machines (default) ● Big 	Yes	Yes

Float Datatype Properties

The standard IEEE 4-byte floating point format.

Table 41. Float Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Scaling Factor (10^n)	<p>This is a value from -16 (shifts the decimal 16 places to the right, making the number smaller by n orders of magnitude) to 0 (keeps the number exactly as it appears in the input) to 16 (shifts the decimal 16 places to the left, making the number larger by n orders of magnitude). To enter a scaling factors from 10^{-16} to 10^{16}, just enter the exponent value.</p> <p>Example: Entering an exponent of 3 will cause any numbers to be multiplied by 1000 (10^3) before being output to XML.</p>	Yes	Yes
Endian	<p>When binary numbers are stored, they can be stored with the smallest component first (little-endian) or last (big-endian).</p> <ul style="list-style-type: none"> ● Little – the standard for Intel x86-based machines (default) ● Big 	Yes	Yes

Integer Datatype Properties

A four-byte integer with a range of 0 to 4,294,967,295 for unsigned or -2,147,483,648 to 2,147,483,647 for signed.

Table 42. Integer Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Signed	<ul style="list-style-type: none"> ● true (default) ● false – use this to make the value unsigned 	Yes	Yes
Scaling Factor (10^n)	<p>This is a value from -16 (shifts the decimal 16 places to the right, making the number smaller by n orders of magnitude) to 0 (keeps the number exactly as it appears in the input) to 16 (shifts the decimal 16 places to the left, making the number larger by n orders of magnitude). To enter a scaling factors from 10^{-16} to 10^{16}, just enter the exponent value.</p> <p>Example: Entering an exponent of 3 will cause any numbers to be multiplied by 1000 (10^3) before being output to XML.</p>	Yes	Yes
Endian	<p>When binary numbers are stored, they can be stored with the smallest component first (little-endian) or last (big-endian).</p> <ul style="list-style-type: none"> ● Little – the standard for Intel x86-based machines (default) ● Big 	Yes	Yes

Long Datatype Properties

An eight-byte integer with a range of 0 to 18,446,744,073,709,551,615 for unsigned or -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 for signed.

Table 43. Long Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Signed	<ul style="list-style-type: none"> ● true (default) ● false – use this to make the value unsigned 	Yes	Yes
Scaling Factor (10^n)	<p>A value from -16 (shifts the decimal 16 places to the right, making the number smaller by n orders of magnitude) to 0 (keeps the number exactly as it appears in the input) to 16 (shifts the decimal 16 places to the left, making the number larger by n orders of magnitude). To enter a scaling factors from 10^{-16} to 10^{16}, just enter the exponent value.</p> <p>Example: Entering an exponent of 3 will cause any numbers to be multiplied by 1000 (10^3) before being output to XML.</p>	Yes	Yes
Endian	<p>When binary numbers are stored, they can be stored with the smallest component first (little-endian) or last (big-endian).</p> <ul style="list-style-type: none"> ● Little – the standard for Intel x86-based machines (default) ● Big 	Yes	Yes

Number Datatype Properties

This corresponds to numbers stored in text form.

Note that if the number is followed by a percent ("%") it is divided by 100; if followed by a permille ("‰") it is divided by 1000.

A leading or trailing '+' is ignored, as numbers are assumed positive. A leading or trailing '-' will make the number negative.

Table 44. Number Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Decimal	The character that corresponds to the decimal point. It is typically a period (".") or comma (","). The default value is determined by the settings on the machine creating the custom XML converter.	Yes	Yes
Thousands	The character that corresponds to the thousands separator. The default is fetched from the system settings, and is typically a comma (","), period (".") or space, and is subsequently stored with the custom XML converter. If seen in the input, it is thrown away and not preserved in the output.	Yes	Yes
Base	The numeric base in which the number is stored in the input file. It can be anything from 2 (binary) to base 36. The default is 10 (decimal). Other common numbers are 8 (octal) and 16 (hexadecimal).	Yes	Yes

Table 44. Number Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Scaling Factor (10^n)	<p>A value from -16 (shifts the decimal 16 places to the right, making the number smaller by n orders of magnitude) to 0 (keeps the number exactly as it appears in the input) to 16 (shifts the decimal 16 places to the left, making the number larger by n orders of magnitude). To enter a scaling factors from 10⁻¹⁶ to 10¹⁶, just enter the exponent value.</p> <p>Example: Entering an exponent of 3 will cause any numbers to be multiplied by 1000 (10³) before being output to XML.</p>	Yes	Yes
'C' Rules for Octal and Hex	<ul style="list-style-type: none"> ● yes – if a number begins with 0 it is considered octal, or if it begins with 0x it is considered hexadecimal; otherwise it is considered decimal (default) ● no – numbers are considered in the given Base only 	Yes	Yes

Table 44. Number Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Use Currency Conventions	<ul style="list-style-type: none"> ● yes – if the number is surrounded by '(' and ')' or has a trailing 'cr' (case-insensitive) it is considered negative, or if it has a trailing 'db' or 'dr' (also case-insensitive) it is considered positive (default) ● no 	Yes	Yes
Left Padding, Right Padding	<p>These two properties control what sort of padding is to be found on either side of the field and then removed. Any characters listed here will be removed. Characters can be specified either by their Unicode values, their Unicode names, or single-quoted.</p> <p>The default for both is spaces (0x20) and tabs (0x09).</p> <p>If there are field delimiters, these are the characters that are trimmed from the contents within the delimiters. The region-level Characters to Toss setting determines which characters are trimmed outside of the delimiters. If there are no delimiters, then the Characters to Toss setting is applied first, and then Left Padding and Right Padding are applied.</p>	Yes	Yes

Short Datatype Properties

A two-byte integer with a range of 0 to 65,535 for unsigned or -32,768 to 32,767 for signed.

Table 45. Short Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Signed	<ul style="list-style-type: none"> ● true (default) ● false – use this to make the value unsigned 	Yes	Yes
Scaling Factor (10^n)	<p>A value from -16 (shifts the decimal 16 places to the right, making the number smaller by n orders of magnitude) to 0 (keeps the number exactly as it appears in the input) to 16 (shifts the decimal 16 places to the left, making the number larger by n orders of magnitude). To enter a scaling factors from 10⁻¹⁶ to 10¹⁶, just enter the exponent value.</p> <p>Example: Entering an exponent of 3 will cause any numbers to be multiplied by 1000 (10³) before being output to XML.</p>	Yes	Yes
Endian	<p>When binary numbers are stored, they can be stored with the smallest component first (little-endian) or last (big-endian).</p> <ul style="list-style-type: none"> ● Little – the standard for Intel x86-based machines (default) ● Big 	Yes	Yes

String Datatype Properties

A regular string of characters.

Table 46. String Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Normalize White Space	<ul style="list-style-type: none"> yes – change all linefeeds, carriage returns and tabs into spaces, then remove all spaces from start and end of text, then collapse any consecutive spaces into a single space (default) no (leave all whitespace alone) 	Yes	Yes
Left Padding, Right Padding	<p>These two properties control what sort of padding is to be found on either side of the field and then removed. Any characters listed here will be removed. Characters can be specified either by their Unicode values, their Unicode names, or single-quoted.</p> <p>The default for both is spaces (0x20) and tabs (0x09).</p> <p>If there are field delimiters, these are the characters that are trimmed from the contents within the delimiters. The region-level Characters to Toss setting determines which characters are trimmed outside of the delimiters. If there are no delimiters, then the Characters to Toss setting is applied first, and then Left Padding and Right Padding are applied.</p>	Yes	Yes

Time Datatype Properties

A time value.

The parsing rules are as follows:

1. If the first character is an 'n' or 'N', it is assumed to be noon (12:00:00).
2. If the first character is an 'm' or 'M', it is assumed to be midnight (00:00:00).
3. If there is a HMS separator, the next three steps are skipped.
4. If the length is six or more, then the hours are the first two characters, the minutes are the next two characters, and the seconds are the next two digits.
5. If the length is four or five, then the hours are the first two characters, and the minutes are the next two characters.
6. No valid date can be determined.
7. If an HMS separator was seen, then the following rules are tried:
8. Up to two digits become the hour, terminated by the HMS separator.
9. Up to two more digits become the minute, also terminated by the HMS separator.
10. If there are any more digits, the next two become the seconds value.
11. If there is a decimal character and more digits, they become fractional parts of the seconds.
12. If an 'am' or 'pm' (case-insensitive) marker is found, the hours are adjusted accordingly.

13. Wrapping is performed for seconds ≥ 60 and minutes ≥ 60 . Hours are moduloed with 24.

Table 47. Time Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
HMS Separator	This is a hint to the parser as to the most likely separator between the hours, minutes and seconds that will be seen. The default value comes from the local system configuration, and also includes the colon (":"), period (".") and space.	Yes	Yes
Left Padding, Right Padding	These two properties control what sort of padding is to be found on either side of the field and then removed. Any characters listed here will be removed. Characters can be specified either by their Unicode values, their Unicode names, or single-quoted. The default for both is spaces (0x20) and tabs (0x09). If there are field delimiters, these are the characters that are trimmed from the contents within the delimiters. The region-level Characters to Toss setting determines which characters are trimmed outside of the delimiters. If there are no delimiters, then the Characters to Toss setting is applied first, and then Left Padding and Right Padding are applied.	Yes	Yes

Zoned Datatype Properties

Also known as the "IBM Signed" format. It is related to Comp3.

Table 48. Zoned Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Scaling Factor (10^n)	A value from -16 (shifts the decimal 16 places to the right, making the number smaller by n orders of magnitude) to 0 (keeps the number exactly as it appears in the input) to 16 (shifts the decimal 16 places to the left, making the number larger by n orders of magnitude). To enter a scaling factors from 10^{-16} to 10^{16} , just enter the exponent value. Example: Entering an exponent of 3 will cause any numbers to be multiplied by 1000 (10^3) before being output to XML.	Yes	Yes

Specifying Control Characters

Lists of symbols are used to designate characters for properties like Field Component Separator, Line Terminator, and Field Separator.

Individual symbols can be expressed in several forms. Numbers and letters, for example, can be entered using single quotes, such as 'A', 'B', 'C'. Control characters or other Unicode values can be expressed using their decimal value, or their hex value by preceding it with 0x – 128 or 0x80, for example. In addition, certain control characters have alternate mnemonic representations, and Stylus Studio supports them, as well.

When entering multiple symbols for a given property, separate symbols using a comma.

The following table summarizes commonly used control characters. For more information, visit <http://www.unicode.org>.

Table 49. Commonly Used Control Characters

<i>Decimal</i>	<i>Hex</i>	<i>Mnemonic</i>	<i>Control Key</i>
0	0x00	NUL	^@, '\0'
1	0x01	SOH	^A
2	0x02	STX	^B
3	0x03	ETX	^C
4	0x04	EOT	^D
5	0x05	ENQ	^E
6	0x06	ACK	^F
7	0x07	BEL or BELL	^G, '\a'
8	0x08	BS	^H, '\b'
9	0x09	TAB or HT	^I, '\t'
10	0x0A	LF	^J, '\n'
11	0x0B	VT	^K, '\v'
12	0x0C	FF	^L, '\f'
13	0x0D	CR	^M, '\r'
14	0x0E	SO	^N
15	0x0F	SI	^O
16	0x10	DLE	^P
17	0x11	DC1 or XON	^Q
18	0x12	DC2	^R
19	0x13	DC3 or XOFF	^S
20	0x14	DC4	^T
21	0x15	NAK	^U

Table 49. Commonly Used Control Characters

<i>Decimal</i>	<i>Hex</i>	<i>Mnemonic</i>	<i>Control Key</i>
22	0x16	SYN	^V
23	0x17	ETB	^W
24	0x18	CAN	^X
25	0x19	EM	^Y
26	0x1A	SUB	^Z
27	0x1B	ESC	^[
28	0x1C	FS	^\
29	0x1D	GS	^]
30	0x1E	RS	^^
31	0x1F	US	^_
127	0x7F	DEL	
128	0x80		
129	0x81		
130	0x82	BPH	
131	0x83	NBH	
132	0x84	IND	
133	0x85	NEL	
134	0x86	SSA	
135	0x87	ESA	
136	0x88	HTS	
137	0x89	HTJ	
138	0x8A	VTB	
139	0x8B	PLD	

Table 49. Commonly Used Control Characters

<i>Decimal</i>	<i>Hex</i>	<i>Mnemonic</i>	<i>Control Key</i>
140	0x8C	PLU	
141	0x8D	RI	
142	0x8E	SS2	
143	0x8F	SS3	
144	0x90	DCS	
145	0x91	PU1	
146	0x92	PU2	
147	0x93	STS	
148	0x94	CCH	
149	0x95	MW	
150	0x96	SPA	
151	0x97	EPA	
152	0x98	SOS	
153	0x99		
154	0x9A	SCI	
155	0x9B	CSI	
156	0x9C	ST	
157	0x9D	OSC	
158	0x9E	PM	
159	0x9F	APC	
160	0xA0	NBSP	
173	0xAD	SHY	

Chapter 4 **Working with XSLT**

Stylus Studio provides many features for creating, updating, and applying stylesheets.

This section of the documentation covers the following topics:

- [“Getting Started with XSLT”](#) on page 325
- [“Tutorial: Understanding How Templates Work”](#) on page 349
- [“Working with Stylesheets”](#) on page 360
- [“Specifying Extension Functions in Stylesheets”](#) on page 377
- [“Working with Templates”](#) on page 382
- [“Using Third-Party XSLT Processors”](#) on page 387
- [“Validating Result Documents”](#) on page 392
- [“Post-processing Result Documents”](#) on page 393
- [“Generating Formatting Objects”](#) on page 394
- [“Generating Scalable Vector Graphics”](#) on page 400
- [“Generating Java Code for XSLT”](#) on page 401
- [“XSLT Instructions Quick Reference”](#) on page 408

Getting Started with XSLT

This section provides an introduction to using Extensible Stylesheet Language Transformations (XSLT). It discusses the following topics:

- [“What Is XSLT?”](#) on page 326
- [“What Is a Stylesheet?”](#) on page 327
- [“What Is a Template?”](#) on page 330

- “How the XSLT Processor Applies a Stylesheet” on page 333
- “Controlling the Contents of the Result Document” on page 339
- “Specifying XSLT Patterns and Expressions” on page 341
- “Frequently Asked Questions About XSLT” on page 343
- “Sources for Additional XSLT Information” on page 344
- “Benefits of Using Stylus Studio” on page 345

What Is XSLT?

The Extensible Stylesheet Language (XSL) is the World Wide Web Consortium's (W3C) language for manipulating XML data. XSLT is the component of XSL that allows you to write a stylesheet that you can apply to XML documents. The result of applying a stylesheet is that the XSLT processor creates a new XML, HTML, or text document based on the source document. The XSLT processor follows the instructions in the stylesheet. The instructions can copy, omit, and reorganize data in the source document, as well as add new data.

XSL is an XML-based language. It was developed by the W3C XSL working group within the W3C Stylesheets Activity. The W3C activity group has organized its specification of XSL into three parts:

- XPath specifies the syntax for patterns and expressions used in stylesheets. The XSLT processor uses an XPath expression to execute a query on the source document to determine which nodes to operate on. See [Writing XPath Expressions](#) on page 629.
- XSLT specifies the syntax for a stylesheet that you apply to one XML document to create a new XML, HTML, or text document.
- XSL formatting object language is an XML vocabulary for specifying formatting instructions.

What XSLT Versions Does Stylus Studio Support?

Stylus Studio 2007 supports XSLT 1.0 and XSLT 2.0. The Stylus Studio built-in XSLT processor is version aware, as is the XSLT Mapper. XSLT 2.0 was designed to work with XPath 2.0.

For more information on

- XSLT 1.0, go to <http://www.w3.org/TR/xslt>
- XSLT 2.0, go to <http://www.w3.org/TR/xslt/20>

To learn more about the changes from XSLT 1.0 to XSLT 2.0, go to <http://www.w3.org/TR/xslt20/#changes>.

What Is a Stylesheet?

A *stylesheet* is an XML document that contains instructions for generating a new document based on information in the source document. This can involve adding, removing, or rearranging nodes, as well as presenting the nodes in a new way.

The following topics provide more information:

- “[Example of a Stylesheet](#)” on page 327
- “[About Stylesheet Contents](#)” on page 330

Example of a Stylesheet

When you work with a stylesheet, three documents are involved:

- XML source document
- Result document, which can be HTML, XML, or text
- XSL stylesheet, which is also an XML document

For example, suppose you have the following XML document:

```
<?xml version="1.0"?>
<bookstore>
  <book>
    <author>W. Shakespeare</author>
    <title>Hamlet</title>
    <published>1997</published>
    <price>2.95</price>
  </book>
  <book>
    <author>W. Shakespeare</author>
    <title>Macbeth</title>
    <published>1989</published>
    <price>9.95</price>
  </book>
  <book>
    <author>D. Alighieri</author>
    <title>The Divine Comedy</title>
    <published>1321</published>
    <price>5.95</price>
  </book>
</bookstore>
```

You can use a stylesheet to transform this XML document into an HTML document that appears as follows in a Web browser:

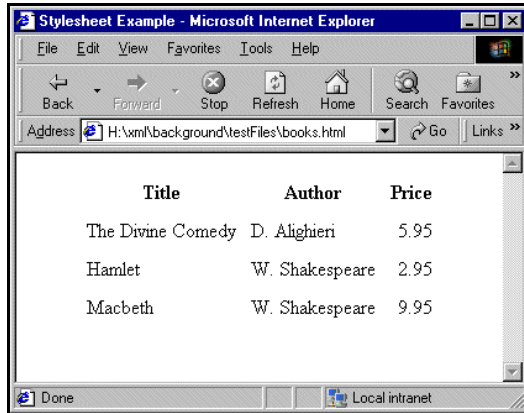


Figure 189. Example of Transformed XML

The Web page in [Figure 189](#) is defined by the following HTML document:

```
<html> <head> <title>Stylesheet Example</title> </head>
<body> <table align="center" cellpadding="5">
<tr><th>Title</th><th>Author</th><th>Price</th></tr>
<tr><td>The Divine Comedy</td><td>D. Alighieri</td>
  <td align="right">5.95</td></tr>
<tr><td>Hamlet</td><td>W. Shakespeare</td>
  <td align="right">2.95</td></tr>
<tr><td>Macbeth</td><td>W. Shakespeare</td>
  <td align="right">9.95</td></tr>
</table> </body> </html>
```

The HTML document contains HTML markup that is not in the source document. In the HTML document, the data from the source document is not in the same order as it is in the XML source document. Also, this HTML document does not include some data that is in the XML source document. Specifically, the HTML document does not include information about the date of publication (the published elements).

To create this HTML file, the stylesheet contains two templates that provide instructions for

- Adding a table with a heading row
- Wrapping the contents of the title, author, and price elements in table cells

Following is a stylesheet that does this.

`xmlns:xsl` is an XSLT instruction. It must be the root element in a stylesheet in Stylus Studio.

`xsl:template` is an XSLT instruction. It contains literal data to be copied to the result document and XSLT instructions to be followed by the XSLT processor. The processor performs these steps for the source nodes identified by the `match` attribute value. In this template, the `match` attribute identifies the root node of the source document.

```
<?xml version = "1.0">
<xsl:stylesheet xmlns:xsl=
  "http://www.w3.org/1999/XSL/Transform"
  version="1.0">
<xsl:output method="html"/>
<xsl:template match="/">
  <html> <head>
    <title>Stylesheet Example</title></head>
    <body>
      <table align="center" cellpadding="5">
        <tr>
          <th>Title</th>
          <th>Author</th>
          <th>Price</th></tr>
          <xsl:apply-templates
            select="/bookstore/book">
            <xsl:sort select="author"/>
          </xsl:apply-templates>
        </table>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="book">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="author"/></td>
      <td align="right">
        <xsl:value-of select="price"/>
      </td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

This template matches book elements in the source document. That is, the template's `match` attribute identifies book elements. In this stylesheet, the XSLT processor performs the actions in this template three times, once for each book element in the source document.

`xsl:value-of` is an XSLT instruction. The XSLT processor extracts the contents of the source node specified in the `select` attribute and copies it into the result document.

Namespace declaration for W3C XSLT namespace.

`xsl:output` is an XSLT instruction. In this stylesheet, it specifies that the result document will be in HTML

`xsl:apply-templates` is an XSLT instruction. For each node identified by this instruction's `select` attribute, the XSLT processor goes to another template in this stylesheet, and performs the actions defined in that template. When done, the processor returns here, and moves to the next line in this template. In this template, the `select` attribute identifies all book elements in the source document.

`xsl:sort` is an XSLT instruction. The XSLT processor processes the book nodes in alphabetical order by author.

About Stylesheet Contents

Stylesheets are XML documents. They contain a combination of

- XSLT elements and attributes. In the previous stylesheet, the XSLT elements are
 - “[xsl:stylesheet](#)” on page 442
 - “[xsl:output](#)” on page 433
 - “[xsl:template](#)” on page 442
 - “[xsl:apply-templates](#)” on page 410
 - “[xsl:sort](#)” on page 439
 - “[xsl:value-of](#)” on page 445

Each XSLT element is an instruction to the XSLT processor. For information about all XSLT instructions, see “[XSLT Instructions Quick Reference](#)” on page 408.

- Non-XSLT elements and attributes. In the previous stylesheet, these include the HTML elements that create the table.

The root element of a stylesheet must declare a namespace that associates a prefix with the URI for an XSLT processor. The URI in the namespace declaration in the previous example identifies the W3C standard XSLT processor. This declaration, shown again below, instructs the XSLT processor to recognize the XSLT elements and attributes by their `xs1` prefix:

```
xmlns:xs1="http://www.w3.org/1999/XSL/Transform"
```

In this stylesheet, you must use the `xs1` prefix for all XSLT instructions.

Note The Stylus Studio XSLT processor requires the namespace URI to be `http://www.w3.org/1999/XSL/Transform`. The prefix can be anything you want. Typically, it is `xs1`.

When you write a stylesheet, you specify the actions you want the XSLT processor to perform when it processes a particular source node. To do this, you define XSLT *templates*, which are described in the next section.

What Is a Template?

A *template* defines what the XSLT processor should do when it processes a particular node in the XML source document. The XSLT processor populates the result document by *instantiating* a sequence of templates. Instantiation of a template means that the XSLT processor

- Copies any literal data from the template to the result document
- Executes the XSLT instructions in the template

The following topics further describe what a template is:

- [Contents of a Template](#) on page 331
- [Determining Which Template to Instantiate](#) on page 332
- [How the select and match Attributes Are Different](#) on page 333

Contents of a Template

The [stylesheet example](#) in “[Example of a Stylesheet](#)” on page 327 defines the following templates using the `xsl:template` instruction:

```
<xsl:template match="/">
  <html><head><title>Stylesheet Example</title></head>
  <body>
  <table align="center" cellpadding="5">
  <tr><th>Title</th><th>Author</th><th>Price</th></tr>
  <xsl:apply-templates select="/bookstore/book">
    <xsl:sort select="author">
  </xsl:apply-templates>
  </table></body></html>
</xsl:template>
<xsl:template match="book">
  <tr><td><xsl:value-of select="title"/></td>
  <td><xsl:value-of select="author"/></td>
  <td align="right"><xsl:value-of select="price"/></td></tr>
</xsl:template>
```

In the `xsl:template` tag, the value of the `match` attribute is an XPath pattern. This pattern matches (identifies) a node or a set of nodes in the source XML document. The value of the `match` attribute is the *template rule*.

The template body defines actions you want the XSLT processor to perform each time it instantiates this template. It contains

- XSLT instructions you want the XSLT processor to follow; for example, `xsl:apply-templates` in the first template, and `xsl:value-of` in the second template.
- Elements that specify literal output you want the XSLT processor to insert in the result document. For example:

```
<table align="center" cellpadding="5">
```

Determining Which Template to Instantiate

When the XSLT processor applies a stylesheet to an XML document, it begins processing with the root node of the XML source document. To process the root node, the XSLT processor searches the stylesheet for a template rule that matches the root node. A template rule matches the root node when the value of the template's match attribute is "/".

If you explicitly defined a template rule that matches the root node, the XSLT processor finds it and instantiates its template. If the XSLT processor does not find an explicitly defined template rule that matches the root node, the processor instantiates the default template that matches the root node. Every stylesheet includes this default template.

Note Whether or not you explicitly define a template rule that matches the root node, the XSLT processor always instantiates a template that matches the root node.

In the [sample stylesheet](#) on “[Example of a Stylesheet](#)” on page 327, the template rule in the first template matches the root node:

```
<xsl:template match="/">
```

The XSLT processor instantiates this template to start generating the result document. It copies the first few lines from the template to the result document. Then the XSLT processor reaches the following XSLT instruction:

```
<xsl:apply-templates select="/bookstore/book"/>
```

When the XSLT processor reaches the select attribute, it creates a list of all source nodes that match the specified pattern. In this example, the list contains book elements. The processor then processes each node in the list in turn by instantiating its matching template. First, the XSLT processor searches for a template that matches the first book element. The template rule in the second template matches the book element:

```
<xsl:template match="book">
```

After instantiating this template for the first book element, the XSLT processor searches for a template that matches the second book element. The XSLT processor instantiates the book template again, and then repeats the process for the third book element. That is, the XSLT processor searches for a matching template, and instantiates that template when it is found.

After three instantiations of the book template, the XSLT processor returns to the first template (the template that matches the root node) and continues with the line after the xsl:apply-templates instruction.

How the select and match Attributes Are Different

Consider the following instructions:

```
<xsl:apply-templates select=expression/>
<xsl:template match=pattern/>
```

The `xsl:apply-templates` instruction uses the `select` attribute to specify an XPath *expression*. The `xsl:template` instruction uses the `match` attribute to specify an XPath *pattern*.

When the XSLT processor reaches an expression that is the value of a `select` attribute, it evaluates the expression relative to the current node. The result of the evaluation is that the XSLT processor selects a set of nodes to be processed.

When the XSLT processor reaches a pattern that is the value of a `match` attribute, it evaluates the pattern alone. The result of the evaluation is that the XSLT processor determines whether or not the pattern matches the node already selected for processing.

For example, suppose you have the following instruction:

```
<xsl:apply-templates select="/bookstore/book"/>
```

This instruction selects the book elements for processing. For each book element, the XSLT processor searches for a template that matches the book element. The following template matches the book element because the pattern identifies all elements that contain author elements. Because book elements contain author elements, this template is a match:

```
<xsl:template match="*[author]">
  <td><xsl:value-of select="author"/></td>
</xsl:template>
```

This example shows that the expression that the XSLT processor uses to select nodes and the pattern it uses to match nodes are independent of each other.

How the XSLT Processor Applies a Stylesheet

When the XSLT processor applies a stylesheet, it starts by automatically selecting the root node for processing and then searching for a template that matches the root node. The XSLT processor then iterates through the process of instantiating templates, selecting nodes in the source document for processing, and matching patterns, until no more templates need to be instantiated.

This section uses the [sample stylesheet](#) on “[Example of a Stylesheet](#)” on page 327 to present this process in more detail in the following topics:

- [Instantiating the First Template](#) on page 334
- [Selecting Source Nodes to Operate On](#) on page 335
- [Controlling the Order of Operation](#) on page 336
- [Omitting Source Data from the Result Document](#) on page 337
- [When More Than One Template Is a Match](#) on page 338
- [When No Templates Match](#) on page 338

Instantiating the First Template

To apply a stylesheet, the XSLT processor searches for a template that matches the source document root. The XSLT processor then instantiates the matching template and begins to process it line by line.

The specific processing depends on the contents of the template that matches the root node. The parts of the template include

- XSLT instructions
- Literal result elements
- Literal result text

It is important to understand that the contents of the XML source document do not dictate the order of XSLT processing. The XSLT processor performs only those actions that you specify, and operates on only the source nodes that you select. For example:

```
<xsl:template match="/">
  <html><head><title>Stylesheet Example</title></head>
  <body>
    <table align="center" cellpadding="5">
      <tr><th>Title</th><th>Author</th><th>Price</th></tr>
      <xsl:apply-templates select="/bookstore/book"/>
    </table></body></html>
</xsl:template>
```

This template matches the root node. Consequently, the XSLT processor begins processing by instantiating this template. This means it processes each part of the template in the order in which it appears.

In the preceding example, the XSLT processor first copies the first four lines in the template body directly into the result document. Then it executes the `xsl:apply-templates` instruction. When execution of that instruction is complete, the XSLT

processor continues processing this template with the last line in the template body. After that, processing of this template is complete, and processing of the stylesheet is also complete.

Selecting Source Nodes to Operate On

Aside from the root node, the XSLT processor operates on only those nodes in the source document that are selected as the result of executing an XSLT instruction. In a stylesheet, there are two XSLT instructions that select nodes in the source document for processing:

```
<xsl:apply-templates select = "expression"/>
<xsl:for-each select ="expression">
    template_body
</xsl:for-each>
```

The value of the `select` attribute is an XPath expression. To evaluate this expression, the XSLT processor uses the current source node as the initial context node. This is the node for which the instruction that contains the `select` attribute is being executed. For example, if this instruction is in the template that matches the root node, the root node is the current source node.

In an `xsl:apply-templates` or `xsl:for-each` instruction, the XSLT processor uses the `select` expression you specify plus the current source node to select a set of nodes. By default, the new list of source nodes is processed in document order. However, you can use the `xsl:sort` instruction to specify that the selected nodes are to be processed in a different order. See [“xsl:sort”](#) on page 439.

When the XSLT processor reaches an `xsl:apply-templates` instruction, the XSLT processor processes each node in the list of selected nodes by searching for its matching template and, if a matching template is found, instantiating it. In other words, the XSLT processor instantiates a template for each node if a matching template is found. The matching template might not be the same template for all selected nodes. If the XSLT processor does not find a matching template, it continues to the next selected node.

In an `xsl:for-each` instruction, the XSLT processor instantiates the embedded template body once for each node in the list of selected nodes.

Controlling the Order of Operation

Typically, the template that matches the root node includes an `xmlns:apply-templates` instruction. When the XSLT processor executes the `xmlns:apply-templates` instruction, it performs the following steps:

1. The processor evaluates the expression specified for the `xmlns:apply-templates select` attribute to create a list of the source nodes identified by the expression.
2. For each node in the list, the XSLT processor instantiates the best matching template. (Template properties such as priority and mode allow multiple templates to match the same node.)
3. The processor returns to the template that contains the `xmlns:apply-templates` instruction and continues processing that template at the next line.

It is important to note that in step 2, the matching template might itself contain one or more `xmlns:apply-templates` instructions. As part of the instantiation of the matching template, the XSLT processor searches for a template that matches the nodes identified by the new `xmlns:apply-templates` instruction. In this way, the XSLT processor can descend many levels to complete processing of the first selected node in the initial `xmlns:apply-templates` instruction. The `xmlns:apply-templates` instruction allows you to access any elements in the source document in any order.

Example

The [sample template](#) on “[Instantiating the First Template](#)” on page 334 contains the following `xmlns:apply-templates` instruction:

```
<xmlns:apply-templates select="/bookstore/book"/>
```

The `select` attribute specifies `"/bookstore/book"` as the expression. This selects the set of `book` elements in the source document as the nodes you want to process. For each selected node, the XSLT processor performs the following steps:

1. The XSLT processor searches the stylesheet for a template that matches `"book"`.

- When the XSLT processor finds the template that matches the book element, it instantiates it. The following template matches the book elements selected by the `xsl:apply-templates` instruction:

```
<xsl:template match="book">
  <tr><td><xsl:value-of select="title"/></td>
  <td><xsl:value-of select="author"/></td>
  <td align="right"><xsl:value-of select="price"/></td></tr>
</xsl:template>
```

- The XSLT processor creates an HTML table row and executes the `xsl:value-of` instructions. These instructions insert the values for the matching book's title, author, and price elements into the table.

The XSLT processor repeats this process for each book node. In other words, it instantiates this template three times, once for each book element in the source document.

It is important to note that the XSLT processor does not search for a matching template once and then instantiate that matching template for each selected element. Rather, the XSLT processor performs the search for a matching template for each node selected for processing. For each node selected for processing, the XSLT processor

- Searches for and chooses the best matching template
- Instantiates the chosen template

Another way to control the order of operation is to specify the `xsl:if`, `xsl:choose`, and `xsl:when` instructions. See [“XSLT Instructions Quick Reference”](#) on page 408.

Omitting Source Data from the Result Document

The XSLT processor operates on only those nodes that you specify. If a node in your XML source document is never referenced in a stylesheet, the XSLT processor never does anything with it.

For example, the [sample source XML document](#) on [“Example of a Stylesheet”](#) on page 327 includes more than the title, author, and price for each book. It also includes the year of publication:

```
<book>
  <author>W. Shakespeare</author>
  <title>Hamlet</title>
  <published>1997</published>
  <price>2.95</price>
</book>
```

However, the template that matches the book element does not specify any processing for the published element. Consequently, the published elements do not appear in the result document.

When More Than One Template Is a Match

Sometimes, more than one template matches the node selected by an `xsl:apply-templates` instruction. In this situation, the XSLT processor chooses the best match. Which match is the best match depends on the template's priority, mode, and order in the stylesheet. Priority, mode, and order are template properties that you can set.

- **Priority** – Priority is a numeric value, such as 1, 10, or 99. The higher the numeric value, the higher the template's priority. Priority is a useful way to distinguish the relative importance of two templates.
- **Mode** – A template's mode allows you to define the context in which a given template should be performed. To use the mode attribute, you specify it (`mode="xyz"`, for example) in both the `xsl:template` and `xsl:apply-template` instructions. Once you have specified a mode, the processor applies a template only if the modes match.
- **Order** – If the XSLT processor cannot distinguish the best match among two or more templates, it uses the last matching template that appears in the stylesheet. Thus, you can enforce priority indirectly by the order in which you define the templates within a stylesheet.

For information on specifying these attributes, see [“xsl:template”](#) on page 442 and [“xsl:apply-templates”](#) on page 410.

When No Templates Match

When the XSLT processor cannot find a template that matches a selected node, it uses built-in templates. Every stylesheet includes built-in templates whether or not you explicitly define them.

The XSLT processor supports these built-in templates:

- The following template matches the root node and element nodes and selects all attributes and child nodes for further processing:

```
<xsl:template match="*/">
  <xsl:apply-templates />
</xsl:template>
```


- The following template matches text and attribute nodes. This template copies the value of the text or attribute node to the result document:

```
<xsl:template match="@*|text()">
  <xsl:value-of select="." />
</xsl:template>
```

Although Stylus Studio does not explicitly insert these templates in stylesheets you create with Stylus Studio, they are always present. That is, as specified by the W3C XSLT Recommendation, these templates are always defined, whether or not they are explicitly defined. See [“Using Stylus Studio Default Templates”](#) on page 384.

Controlling the Contents of the Result Document

This section highlights some of the XSLT instructions you can specify in a stylesheet to control the contents of the result document. This section discusses the following topics:

- [Specifying Result Formatting](#) on page 339
- [Creating New Nodes in the Result Document](#) on page 340
- [Controlling White Space in the Result](#) on page 340

Specifying Result Formatting

In a stylesheet, you can specify that the XSLT processor should format the result as XML, HTML, or text. [Table 50](#) describes the XSLT processor output for each alternative:

Table 50. Output Based on Result Format

<i>Result Format</i>	<i>XSLT Processor Output</i>
XML	Well-formed XML.
HTML	Recognized HTML tags and attributes that are formatted according to the HTML 4.0 specification. Most browsers should be able to correctly interpret the result. It is your responsibility to ensure that the result is well-formed HTML. For example, elements should not have child nodes.
Text	All text nodes in the result in document order.

See [“xsl:output”](#) on page 433 for information about specifying formatting in a stylesheet.

Creating New Nodes in the Result Document

The simplest way to create new nodes in a result document is to specify them as literal result elements or literal result text in a stylesheet template. For example:

```
<xsl:template match="/">
  <html><head></head><body><table>
    <tr><th>Title</th><th>Author</th><th>Price</th></tr>
    ...
  </table></body></html>
</xsl-template>
```

This template creates many nodes in the result document that were not in the source document.

You can also use XSLT instructions to create new nodes. Typically, you use XSLT instructions when you need to compute the name or value of the node. You can find information about using the following instructions in the [“XSLT Instructions Quick Reference”](#) on page 408:

- [“xsl:element”](#) on page 421
- [“xsl:attribute”](#) on page 411
- [“xsl:comment”](#) on page 418
- [“xsl:processing-instruction”](#) on page 438
- [“xsl:text”](#) on page 444

You can use the [xsl:value-of](#) on page 445 instruction to provide the contents for a new node. You can also create a new node by copying the current node from the source document to the result document. The current node is the node for which the XSLT processor instantiates a template. See [“xsl:copy”](#) on page 418.

Controlling White Space in the Result

For readability, XML documents (both source documents and stylesheets) often include extra white space. White space in XML documents includes spaces, tabs, and new-line characters. Because this white space is for readability, it receives special treatment.

Text nodes that contain only white space are

- Preserved as normal text nodes in a source document
- Ignored in a stylesheet, unless the parent node is `xsl:text`

Significant white space

Stylus Studio recommends that you specify `xmlns:text` in a stylesheet whenever you want to create significant white space in the result. *Significant white space* is white space that you want to appear in the result in exactly the way that you specify.

To obtain white space for readability during output formatting, specify the `xmlns:output` instruction with the `indent` attribute. Default values are `yes` for HTML, and `no` for XML. With Stylus Studio, you can select the **Indent** check box on the **Params/Other** tab to display indented output instead of one long string. Note that the value of the `indent` attribute, if specified in the stylesheet, has precedence over the **Indent** option.

Specifying XSLT Patterns and Expressions

In a stylesheet's `xmlns:template`, `xmlns:apply-templates`, `xmlns:for-each`, and `xmlns:value-of` instructions, you specify patterns or expressions as the values for the `match` or `select` attributes. These patterns are XPath expressions. You specify patterns or expressions to

- Define which nodes a template rule matches.
- Select lists of source nodes to process.
- Extract source node contents to generate result nodes.

Depending on the context, an XSLT pattern or expression can mean one of the following:

- Does this template match the current node?
- Given the current node, select all matching source nodes.
- Given the current node, select the first matching source node.
- Given the current node, do any source nodes match?

Patterns or expressions can match or select any type of node. The XSLT processor can match a pattern to a node based on the existence of the node, the name of the node, or the value of the node. You can combine patterns and expressions with Boolean operators. For detailed information about patterns and expressions, see [“Writing XPath Expressions”](#) on page 629.

Examples of Patterns and Expressions

Following are examples of patterns and expressions you can specify in stylesheet instructions:

```
xmlns:template match = "book/price"
```

Matches any price element that is a child of a book element.

```
xsl:template match = "book//award"
```

Matches any award element that is a descendant of a book element.

```
xsl:template match = "book [price]"
```

Matches any book element that has a child that is a price element.

```
xsl:template match = "book [@price]"
```

Matches any book element that has a price attribute.

```
xsl:template match = "book [price=14]"
```

Matches any book element that has a child that is a price element whose value is 14.

```
xsl:template match = "book [@price=14]"
```

Matches any book element that has a price attribute whose value is 14.

```
xsl:apply-templates select = "book"
```

Selects all book elements that are children of the current element.

```
xsl:apply-templates select = "book/price"
```

Selects all price elements that are children of book elements that are children of the current element.

```
xsl:apply-templates select = "//book"
```

Selects all book elements in the source document.

```
xsl:apply-templates select = "./book"
```

Selects all book elements that are descendants of the current element.

Frequently Asked Questions About XSLT

How can I use quoted strings inside an attribute value?

If you need to include a quoted string inside an attribute value (in a `select` expression, for example), you can use the single quotation mark character (') in the value of the attribute. For example:

```
select = "book[title = 'Trenton Today']".
```

How do I choose when to use `xsl:for-each` and when to use `xsl:apply-templates`?

The way `xsl:for-each` and `xsl:apply-templates` select nodes for processing is identical. The way these instructions find the templates to process the selected nodes is different.

With `xsl:for-each`, the template to use is fixed. It is the template that is contained in the body of the `xsl:for-each` element. With `xsl:apply-templates`, the XSLT processor finds the template to be used for each selected node by matching that node against the template rules in the stylesheet.

Finding a template by matching requires more time than using the contained template. However, matching allows for more flexibility. Also, matching lets you avoid repeating templates that might be used in more than one place in a stylesheet.

Named templates are another option for invoking a template from more than one place in a stylesheet, when you know which template you want. It is a common mistake to use (and bear the overhead of) matching when it is not needed. But it allows you to do powerful things. Matching can take into account the following:

- Pattern matching on the node
- Precedence of templates based on stylesheet importance
- Template priority
- Template ordering

Most complex document-formatting stylesheets use `xsl:apply-templates` extensively.

Tip Use the XSLT Profiler to help you understand where the processor is spending most of its time. See [“Profiling XSLT Stylesheets”](#) on page 500.



XSLT Profiling is available only in Stylus Studio XML Enterprise Suite.

How can I insert JavaScript in my result document?

If you want your result document to contain JavaScript commands, you must properly escape the JavaScript code. Use the following format in your XSLT template:

```
<script>
  <xsl:comment>
    <![CDATA[ <your JavaScript here> ]]>
  </xsl:comment>
</script>
```

However, this method does not work when your JavaScript section contains a block of XSLT code. In this case, enclosing the JavaScript in a CDATA tag causes the XSLT processor to ignore not just the JavaScript but also the markup code within that tag.

In this situation, enclose the entity reference within an `<xsl:text>` tag with `disable-output-escaping` set to "yes". For example:

```
if(length <xsl:text disable-output-escaping="yes">&gt;</xsl:text> 1)
```

You can use this wherever an entity reference needs to be handled specifically, as opposed to being handled as part of an entire JavaScript section.

My browser does not understand the tag `
`. How can I output just `
`?

Although your XSLT stylesheet must contain valid XML (meaning all tags must be either empty or have a closing element), you can instruct the XSLT processor to generate output compliant with HTML. See “[Deleting Templates](#)” on page 387.

Alternative: To ensure that your stylesheet always generates correct HTML, specify the `xsl:output` instruction with the `method` attribute set to `html`. See “[xsl:output](#)” on page 433.

Sources for Additional XSLT Information

For additional information about XSL and XSLT, visit the following Web sites:

- <http://www.w3.org/Style/XSL/>
W3C Extensible Stylesheet Language specification
- <http://www.w3.org/TR/xslt>
W3C XSLT Recommendation

Benefits of Using Stylus Studio

Now that you have an understanding of what a stylesheet can do, you can appreciate the benefits of using Stylus Studio to create them. Stylus Studio is the first integrated environment for creating, managing, and maintaining an XSL-enabled Web presence. By combining the tools needed to create XSLT stylesheets in a visual editing environment, Stylus Studio speeds initial development and eases maintenance. Key elements of Stylus Studio's XSLT features include

- [Structural Data View](#) on page 345
- [Sophisticated Editing Environment](#) on page 346
- [XSLT and Java Debugging Features](#) on page 347
- [Integrated XML Parser/XSLT Processor](#) on page 349

Structural Data View

Stylus Studio graphically displays the structure, or schema, of the XML data to which you want to apply a stylesheet.

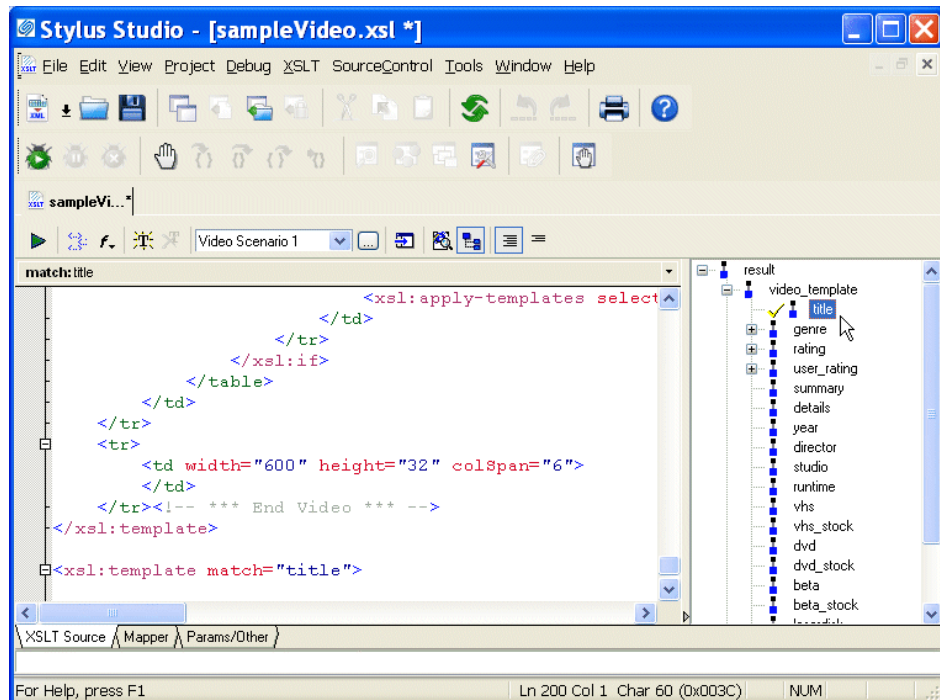


Figure 190. Tree View Lets You Easily Edit XSLT

Using this tree view, you can apply formatting to your XML – double-clicking a node in the tree automatically adds an `xsl:template match=` instruction for that node, for example. Similarly, when you drag a node into the XSLT source, Stylus Studio displays a pop-up menu that allows you to easily insert an XSLT instruction.

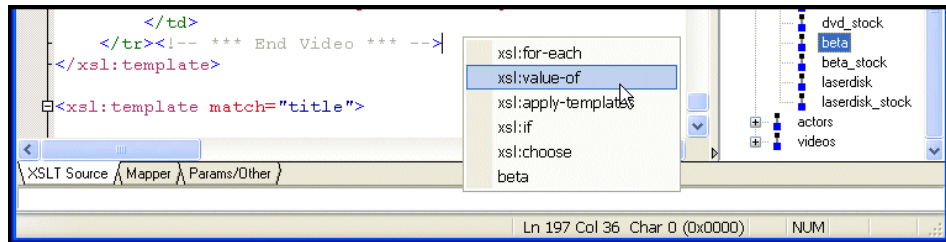


Figure 191. Stylus Studio Displays XSLT Instructions for Quick Editing

Finally, you can also use the tree to move quickly among different XSLT templates – clicking a node in the tree places the cursor at the corresponding template in the XSLT source.

Sophisticated Editing Environment

The Stylus Studio editor allows you to edit both the XML source document and the XSLT stylesheet. There is no need to memorize complicated syntax. As you type, Stylus Studio

Sense:X technology automatically suggests XSLT or HTML tag and attribute names, and ensures that all XML is well formed.

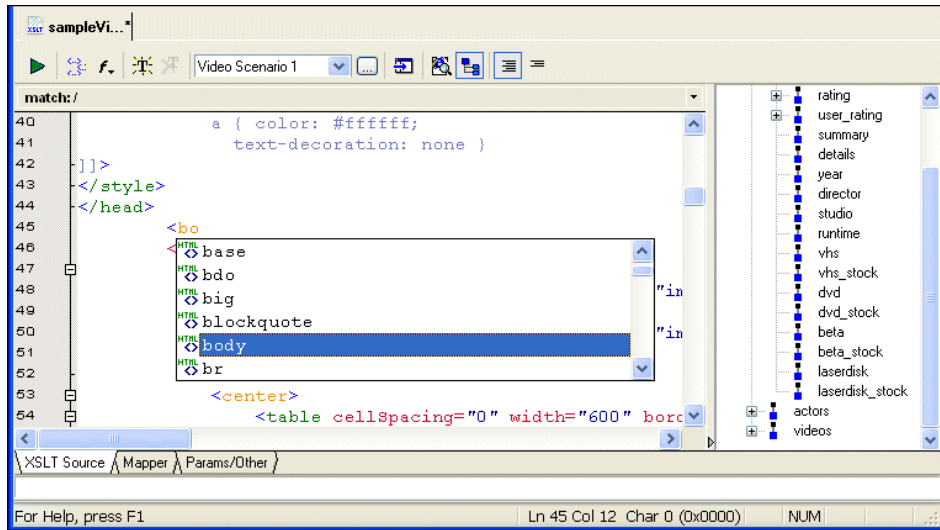


Figure 192. Sense:X Speeds Coding, Reduces Errors

Sense:X also adapts to your document by suggesting more frequently used tags first. Valid XSLT and HTML tag names are color coded to improve readability.

XSLT and Java Debugging Features

Complex stylesheets require robust debugging features. With Stylus Studio, you can do the following:

- Set breakpoints in your stylesheet.
- Monitor the value of XSLT variables.
- Trace the sequence of XSLT instructions that created HTML output. With a click anywhere in the rendered HTML page, Stylus Studio Visual Backmapping

technology displays the XSLT instructions responsible for creating that portion of HTML output.

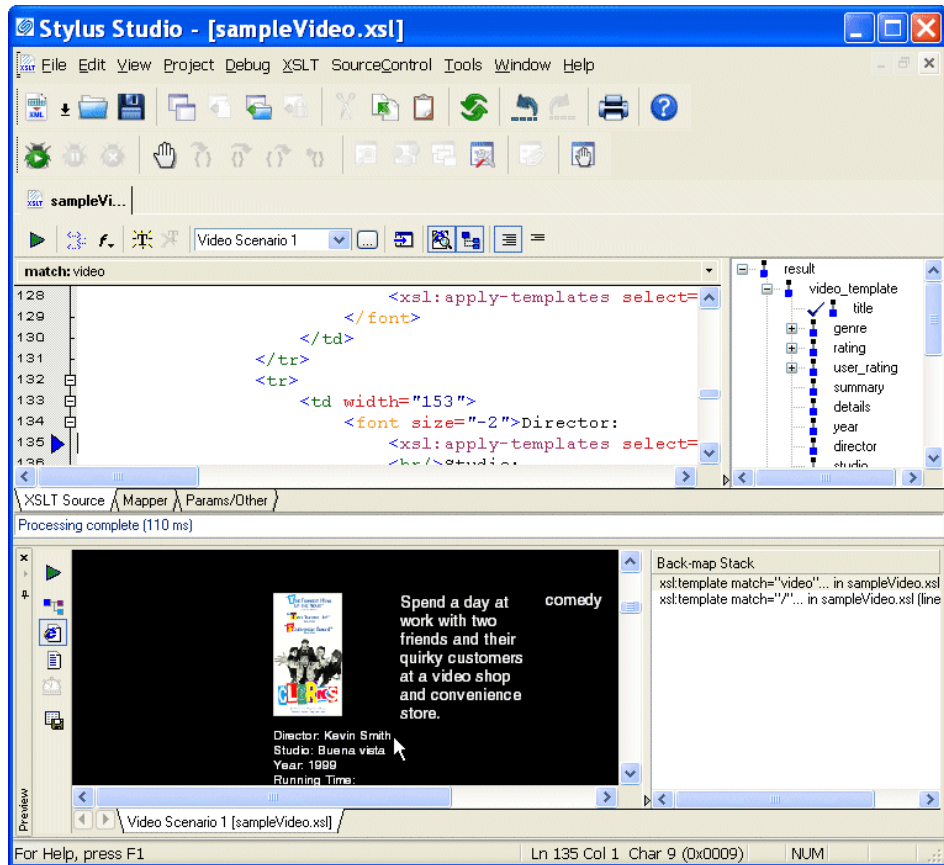


Figure 193. Click to HTML Output to Backmap to XSLT Source

Also, you can click in the stylesheet and the backmapping feature highlights the text generated by that template.

- Use the XSLT Profiling report to review performance metrics to help troubleshoot and tune your XSLT stylesheets.



XSLT Profiling is available only in Stylus Studio XML Professional Suite.

Integrated XML Parser/XSLT Processor

Stylus Studio integrates an XML parser with an XSLT processor. This allows Stylus Studio to instantly show the output of your stylesheet. Each time you apply a stylesheet to an XML document, Stylus Studio detects and flags any errors in your stylesheet or XML data.

Stylus Studio's default XSLT processor is compliant with the W3C XSLT Recommendation. You can also use custom processors of your own.

Tutorial: Understanding How Templates Work

When Stylus Studio creates a new stylesheet, it contains one template, which matches the root node. However, this template is empty. If you apply the new stylesheet as is, the result document has no contents. To generate a result document with contents, you need to add instructions to the template that matches the root node.

All stylesheets have two default templates that do not appear in the stylesheet itself. It is important for you to understand how the default templates work so that you can

- Add instructions to the template that matches the root node.
- Define additional templates to operate on the elements in your document.
- Specify HTML markup in templates.

When you can do this, you can write a stylesheet that generates a dynamic Web page that displays your information.

This tutorial provides step-by-step instructions for defining a stylesheet that generates a dynamic Web page from an XML document. The tutorial shows how the default templates work, and it provides instructions for defining templates that instantiate the default templates. It also provides instructions for adding HTML markup to the stylesheet. The result is a dynamic Web page that displays the particular information you choose.

Each of the following topics contains instructions for defining the stylesheet. You should perform the steps in each topic before you move on to the next topic. After the first topic, some steps depend on actions you performed in a previous topic. This section organizes the process as follows:

- [Creating a New Sample Stylesheet](#) on page 350
- [Understanding How the Default Templates Work](#) on page 353
- [Editing the Template That Matches the Root Node](#) on page 358
- [Creating a Template That Matches the book Element](#) on page 358

- [Creating a Template That Matches the author Element](#) on page 359

For a simpler tutorial that shows you how to define a stylesheet that generates a dynamic Web page from a static HTML document, see “[Working with Stylesheets – Getting Started](#)” on page 27.

This tutorial duplicates some of the information in subsequent sections. For complete information, see the following topics:

- [Working with Stylesheets](#) on page 360
- [Working with Templates](#) on page 382

Creating a New Sample Stylesheet

- ◆ **To create a stylesheet to use in this tutorial, follow these instructions:**

1. From the Stylus Studio menu bar, select **File > New > XSLT Stylesheet**.

Stylus Studio displays a new untitled stylesheet and the **Scenario Properties** dialog box, and selects the text in the **Scenario Name** field.

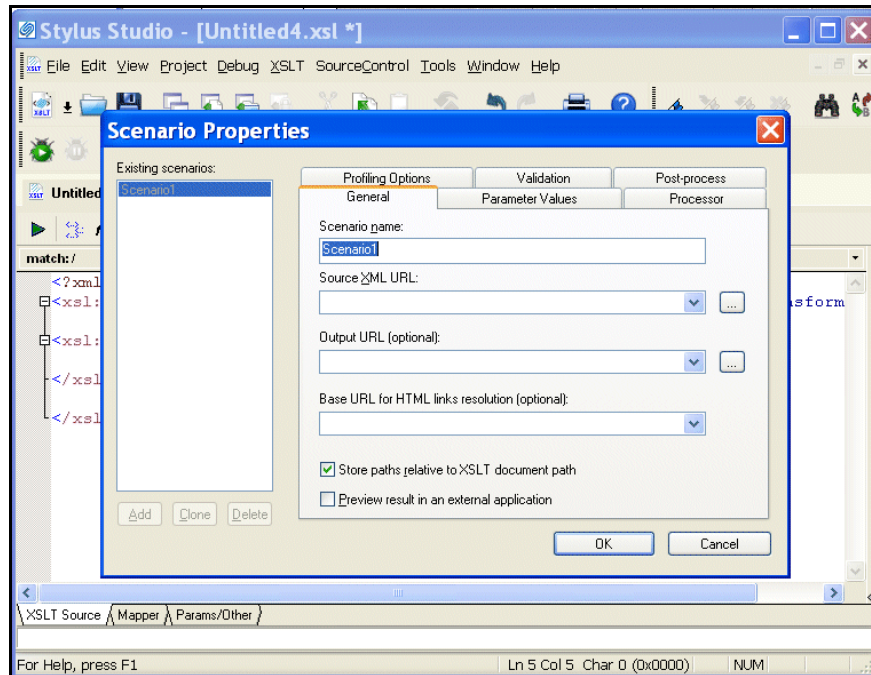


Figure 194. Scenarios Let You Easily Test Different XSLT/XML Pairs

2. In the **Scenario Properties** dialog box, in the **Scenario Name** field, type **DynamicBookstoreScenario**.
3. Click **Browse** to the right of the **Source XML URL:** field. Stylus Studio displays the **Open** dialog box.
4. Navigate to the Stylus Studio examples\query directory.
5. Double-click **bookstore.xml**. This is the XML document that the new stylesheet will operate on.
6. In the **Scenario Properties** dialog box, click **OK**.

This creates a scenario with the name `DynamicBookstoreScenario`. This scenario associates the `bookstore.xml` document with the new stylesheet. If you want to apply the new stylesheet to other XML documents, you must create a new scenario or change the name of the XML document in this scenario.

Stylus Studio displays the new stylesheet in the XSLT editor. A tree representation of the `bookstore.xml` document appears to the right.

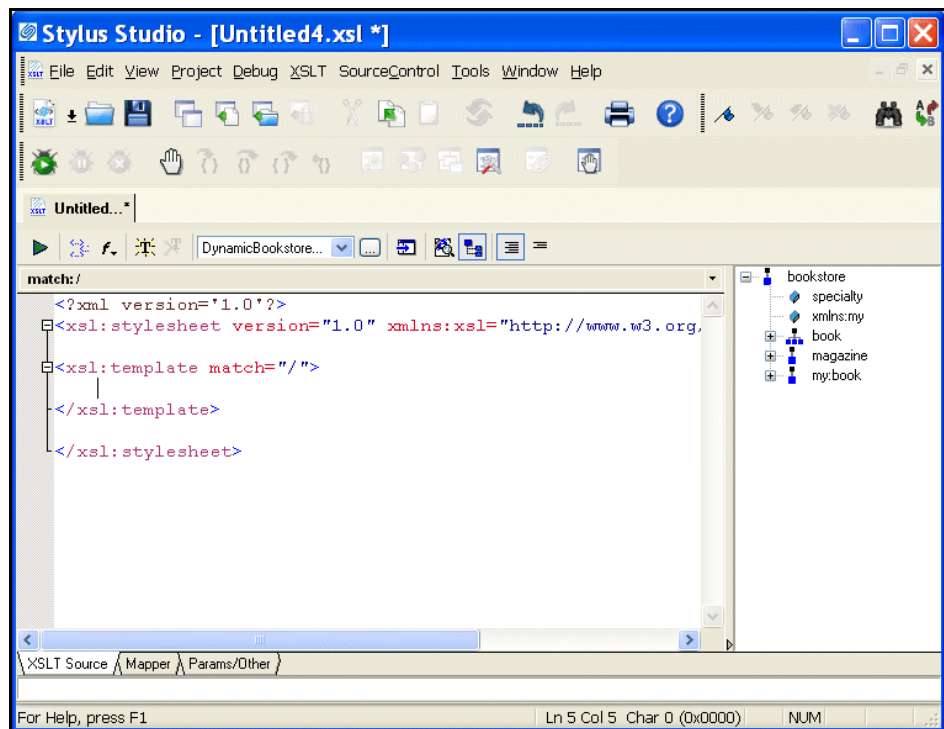




Figure 195. The XSLT Editor Shows XSLT Source on Left, Tree on Right

The default stylesheet that Stylus Studio creates contains one template, which matches the root node.

7. In the XSLT editor tool bar, click **Preview Result**  .
Stylus Studio displays the **Save As** dialog box so you can save the XSLT you are composing.
8. In the **URL:** field, type `myStylesheet.xsl` and click **Save**.
Stylus Studio applies the new stylesheet to `bookstore.xml` and displays the result in the **Preview** window. The result, displayed in the **Preview** window, has no contents because the template that matches the root node is empty.
9. In the XSLT editor pane, click in the empty line that follows `<xsl:template match="/">` .
10. Type `<x`, which displays the Sense:X completion list.
11. In the completion list, scroll down and click **xsl:apply-templates**.
12. Type `/>`.
13. In the XSLT editor tool bar, click **Preview Result**  .
Stylus Studio displays the **Save As** dialog box.
14. Enter a name for the file and click **Save**.

This time, the **Preview** window contains all text in bookstore.xml and none of the markup. This is because the `xsl:apply-templates` instruction instantiates the default templates.

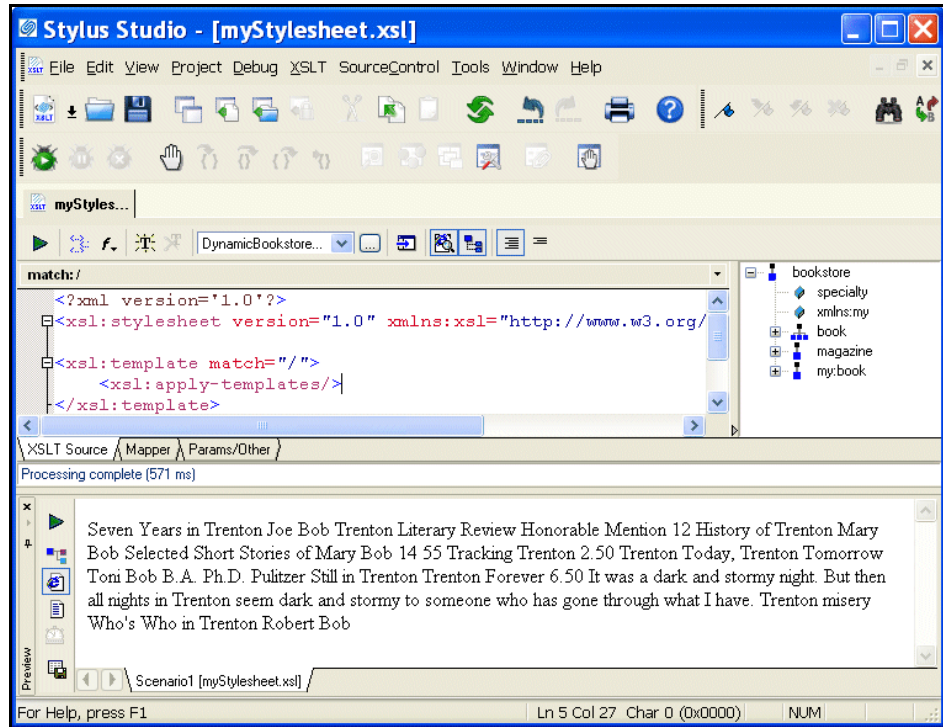


Figure 196. Default Templates Contain No Formatting Instructions

To create a Web page, you need to add HTML markup that displays the information the way you want. To make it easier to do that, you need to understand how the text is already being copied to the result document.

Understanding How the Default Templates Work

After you complete the steps in the previous section, you can see the bookstore.xsl stylesheet in the XSLT editor pane. It has the following contents:

```

<?xml version='1.0' ?>
<xsl:stylesheet version="1.0"xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>
</xsl:stylesheet>

```

The stylesheet explicitly contains one template, which matches the root node. When the XSLT processor applies a stylesheet, the first thing it does is search for a template that matches the root node. If there is no template that explicitly matches the root node, the XSLT processor uses a built-in template.

There are two built-in templates, also called default templates. Every XSLT stylesheet contains these templates whether or not they are explicitly specified. This is part of the W3C XSLT Recommendation.

This section discusses the following topics:

- [Instantiating the Template That Matches the Root Node](#) on page 354
- [Instantiating the Root/Element Default Template](#) on page 355
- [Instantiating the Text/Attribute Default Template](#) on page 356
- [Illustration of Template Instantiations](#) on page 357

Instantiating the Template That Matches the Root Node

The XSLT processor instantiates the template that matches the root node. The template that matches the root node contains only the `xs1:apply-templates` instruction. In this template, the `xs1:apply-templates` instruction does not specify a `select` attribute. Consequently, the XSLT processor operates on the children of the node for which the root template was instantiated. In the `bookstore.xml` document, the root node has three children:

- XML declaration
- Comment
- bookstore document element



Figure 197. Source XML Document from DynamicBookstoreScenario

Unless you specify otherwise, the XSLT processor operates on the children in document order. The first child is a processing instruction (the XML declaration). The XSLT processor ignores processing instructions.

The second child is the comment node, and the XSLT processor also ignores comment nodes.

The third child is the bookstore document element. The XSLT processor searches for a template that matches bookstore. Because there is no template that explicitly matches the bookstore element, the XSLT processor instantiates a built-in template that is not explicitly in the stylesheet.

Instantiating the Root/Element Default Template

One default template matches `*|/`. This means it matches every element in the source document, and it also matches the root node. This is the root/element default template.

The root/element default template contains only the `xmlns:apply-templates` instruction. Like the template that matches the root node, the `xmlns:apply-templates` instruction in the root/element default template does not specify a `select` attribute. That is, it does not identify the set of nodes for which templates should be applied. Consequently, the XSLT processor operates on the children of the node for which the root/element template was instantiated.

In this case, the root/element default template was instantiated for the bookstore element. The children of the bookstore element include four book elements, a magazine element, and a book element associated with the `my` namespace.

The XSLT processor operates on these children in document order. First, it searches for a template that matches `book`. Because there is no template that explicitly matches the book element, the XSLT processor instantiates the root/element default template for the first book element.

Again, by default, the `xmlns:apply-templates` instruction in the root/element default template operates on the children of the current node in document order. That is, it operates on the children of the first book element.

In the first book element, the first child is the `title` element. The XSLT processor searches for a template that matches the `title` element. Because there is no template that explicitly matches the `title` element, the XSLT processor instantiates the root/element default template again.

At this point, the XSLT processor has initiated instantiation of the root template once, and the root/element default template several times:

```
Instantiate root template for root node.  
  Instantiate root/element template for bookstore element.  
    Instantiate root/element template for first book element.  
      Instantiate root/element template for title in first book element.
```

It is important to understand that these instantiations are not yet complete. Each subsequent instantiation of the root/element default template is inside the previous instantiations.

Instantiating the Text/Attribute Default Template

When the XSLT processor instantiates the root/element default template for the `title` element, the `xs1:apply-templates` instruction operates on the children of the `title` element. The `title` element has one child, which is a text node. The XSLT processor searches for a template that matches this text node. The second default template in the stylesheet matches this text node. This template matches `text()|@*`, meaning that it matches every text node and every attribute in the source document. This is the text/attribute template.

The XSLT processor instantiates the text/attribute default template for the `title` element's text node. This template contains only the `xs1:value-of` instruction. Its `select` attribute identifies the current node, which is the node for which the template was instantiated. This template copies the text contained in the current text node to the result document.

Now the result document contains the following text:

```
Seven Years in Trenton
```

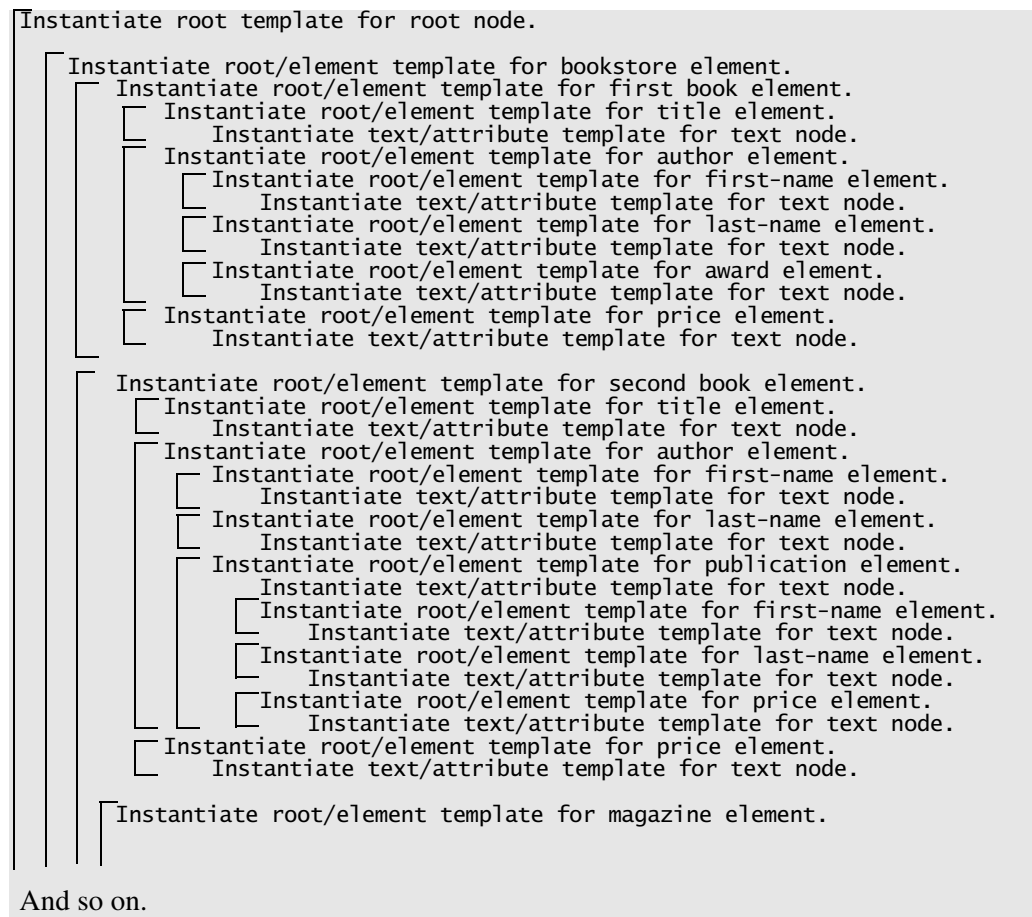
The XSLT processor is finished with the `title` element, and it next processes the `author` element in the first book element. There is no template that explicitly matches `author`, so the XSLT processor instantiates the root/element default template. The first child of the `author` element is the `first-name` element, and again, there is no template that explicitly matches the `first-name` element. The XSLT processor instantiates the root/element default template for the `first-name` element. The only child of the `first-name` element is a text node. The XSLT processor instantiates the text/attribute default template for this text node, and this template copies the text to the result document. Now the result document contains the following text:

```
Seven Years in Trenton Joe
```

The XSLT processor is finished with the first-name element, and it next processes the last-name element, which is the second child of the author element.

Illustration of Template Instantiations

As you can see from the description in the previous section, the XSLT processor iterates through the process of searching for a matching template, instantiating one of the default templates, and operating on the children of the node for which the template was instantiated. The following figure shows the template instantiations through the second book element. In the figure, each bracket encloses the instantiations that together compose a complete instantiation for a particular element.



Editing the Template That Matches the Root Node

Begin writing your stylesheet by adding instructions to the template that explicitly matches the root node in your source document:

In the XSLT editor, edit the contents of the root template so that it contains only the following contents. As you type, Stylus Studio displays a pop-up menu that lists possible instructions. You can scroll the list and double-click the entry you want, or you can continue typing.

Ensure that you do one of the following:

- Remove the `xsl:apply-templates` instruction that you inserted earlier.
- Edit the `xsl:apply-templates` instruction to include the `select` attribute as shown above, and place it in the correct location.

Creating a Template That Matches the `book` Element

The template that matches the root node includes an `xsl:apply-templates` instruction that selects `book` nodes for processing.

◆ **To define the template that matches the `book` element:**

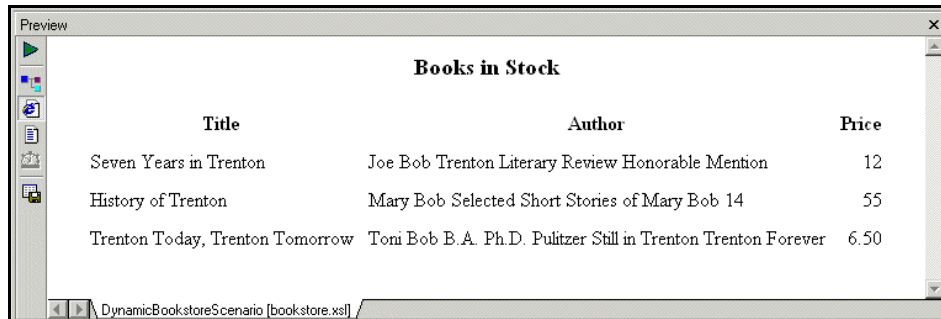
1. In the XSLT editor source document tree pane, expand the **bookstore** element.
2. Double-click the **book** element.

Stylus Studio creates a template that matches the `book` element. The new template is near the end of the stylesheet and has the form `<xsl:template match="book">`. In the tree pane, the yellow check next to the **book** element indicates that there is a template that matches this element.

3. In the XSLT editor pane, add the following instructions to the new template's body:

```
<tr>
  <td><xsl:apply-templates select="title"/></td>
  <td><xsl:apply-templates select="author"/></td>
  <td align="right">
    <xsl:apply-templates select="price"/>
  </td>
</tr>
```

Press F5 to see the results. The result document looks like that shown in [Figure 198](#):



Title	Author	Price
Seven Years in Trenton	Joe Bob Trenton Literary Review Honorable Mention	12
History of Trenton	Mary Bob Selected Short Stories of Mary Bob 14	55
Trenton Today, Trenton Tomorrow	Toni Bob B.A. Ph.D. Pulitzer Still in Trenton Trenton Forever	6.50

Figure 198. Result of Applying XSLT

In the book template, the `xmlns:apply-templates` instructions cause the XSLT processor to instantiate the default templates. For the `title` and `price` elements, this works correctly because those elements include only a text node. But for the `author` element, the use of the default templates copies too much information to the result table. You need to explicitly define a template for the `author` element.

Creating a Template That Matches the `author` Element

◆ **To define a template that matches the `author` element:**

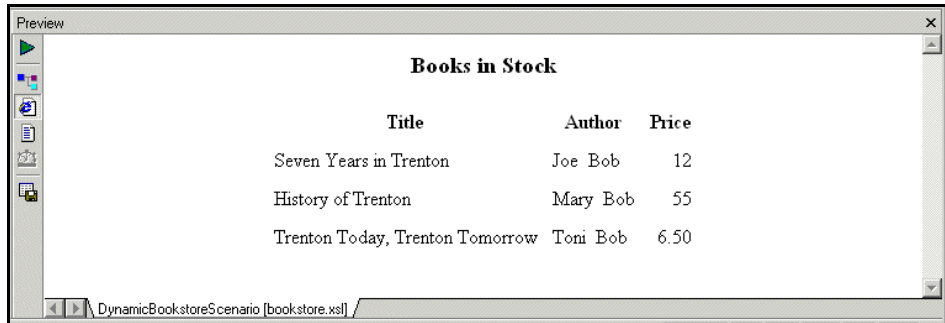
1. In the XSLT editor source document tree pane, expand the **book** element.
2. Double-click the **author** element.

Stylus Studio creates a template that matches the `author` element, and places it near the end of the stylesheet.

3. In the XSLT editor pane view, edit the template body so that it contains only the following contents.

```
<xsl:value-of select="first-name"/>
 
<xsl:value-of select="last-name"/>
```

If you do not include the nonbreaking space entity, the first name and the last name have no space between them. Press F5 to see the results of this change, as shown in [Figure 199](#).



The screenshot shows a 'Preview' window with a table titled 'Books in Stock'. The table has three columns: 'Title', 'Author', and 'Price'. The data rows are as follows:

Title	Author	Price
Seven Years in Trenton	Joe Bob	12
History of Trenton	Mary Bob	55
Trenton Today, Trenton Tomorrow	Toni Bob	6.50

The window title bar shows 'DynamicBookstoreScenario [book.store.xml]'.

Figure 199. Result of XSLT with an Author Template

4. Save the stylesheet by clicking **Save** .
5. Close the stylesheet by clicking **File > Close** on the Stylus Studio menu bar.

Working with Stylesheets

This section provides instructions for performing the various tasks involving stylesheets. See also “[Working with Templates](#)” on page 382. This section covers the following topics:

- [About the XSLT Editor](#) on page 361
- [Creating Stylesheets](#) on page 362
- [Creating a Stylesheet from HTML](#) on page 362
- [Specifying Stylesheet Parameters and Options](#) on page 363
- [Applying Stylesheets](#) on page 366
- [Applying a Stylesheet to Multiple Documents](#) on page 372
- [About Stylesheet Contents](#) on page 373
- [Updating Stylesheets](#) on page 374
- [Saving Stylesheets](#) on page 376

Also, Stylus Studio provides a number of tools that help you debug stylesheets. See “[Debugging Stylesheets](#)” on page 493.

About the XSLT Editor

The XSLT editor, which displays a stylesheet when you open it, has four tabs at the bottom.

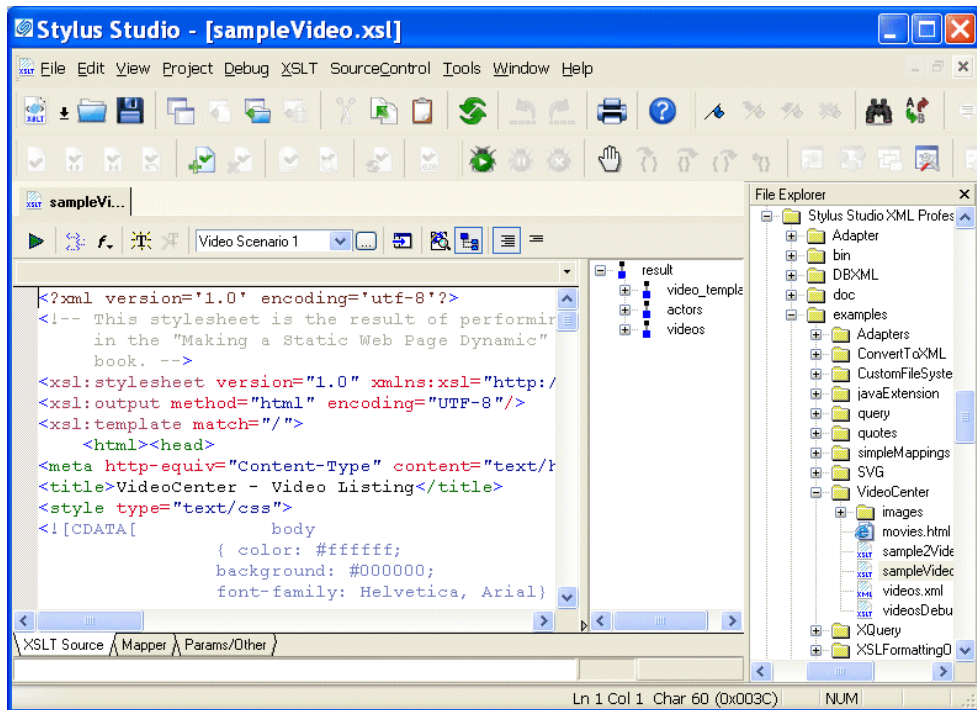


Figure 200. XSLT Editor

The **XSLT Source**, **Mapper**, and **Params/Other** tabs are always available.

Editing XSLT as XML

If you want, you can edit an XSLT file as an XML file. To do this, open the stylesheet in the XML editor instead of in the XSLT editor. In the **Open** dialog box, click the down arrow in the **Open** button. Click **XML Editor** in the drop-down menu. A document can be open in the XML editor and in the XSLT editor at the same time.

Creating Stylesheets

◆ To create a stylesheet:

1. From the Stylus Studio menu bar, select **File > New > XSLT Stylesheet**. Stylus Studio displays the **Scenario Properties** dialog box.
2. In the **Scenario Name:** field, type a name for the association between the new stylesheet and a particular XML source document. You might want to use the convention of specifying the name you want your result document to have. The result document is the document that will contain the result of applying the stylesheet you are about to create.
3. In the **Source XML URL:** field, type the name of an XML document or click **Browse** to navigate to a document. Select a document you want to apply the new stylesheet to. You are not limited to applying the new stylesheet to only this XML document. You can create other scenarios later and specify other XML documents to which you want to apply the same stylesheet.
4. Click **OK**. Stylus Studio displays an untitled stylesheet window. The default text in the new stylesheet appears in the left pane. The schema for the XML source document you specified in the scenario properties appears in the right pane.
5. To give the stylesheet a name, select **File > Save**.
6. Navigate to where you want to save the stylesheet.
7. In the URL: field, type the new stylesheet's name.
8. Click **Save**.

Creating a Stylesheet from HTML

You can create an XSLT stylesheet from an HTML file using the HTML to XSLT document wizard.

Tip Stylus Studio also has a document wizard that converts HTML to XML. See [Creating XML from HTML](#) on page 142.

◆ To run the HTML to XSLT document wizard:

1. Select **File > Document Wizards** from the menu. The **Document Wizards** dialog box appears.
2. Click the **XSLT Editor** tab.

3. Double-click **HTML to XSLT** (or select the **HTML to XSLT** icon and click **OK**).
The **HTML to XML** dialog box appears.

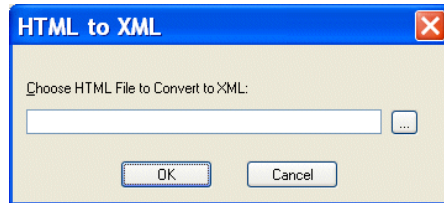



Figure 201. HTML to XML Dialog Box

4. Enter the name of the HTML file you want to convert to XSLT in the **Choose HTML File to Convert to XSLT** field.
5. Click **OK**.
Stylus Studio opens the converted HTML file as an untitled XSLT document in the XSLT Editor.

Specifying Stylesheet Parameters and Options

You can specify values for stylesheet parameters in the **Parameter Values** tab of the **Scenario Properties** dialog box.

◆ To specify XSLT stylesheet parameters:

1. Open the stylesheet for which you want to specify parameter values.
2. In the XSLT editor tool bar, click **Browse**  .
Stylus Studio displays the **Scenario Properties** dialog box.

3. Click the **Parameter Values** tab.

Stylus Studio displays a list of the parameters defined in your stylesheet, if any, with any default values.

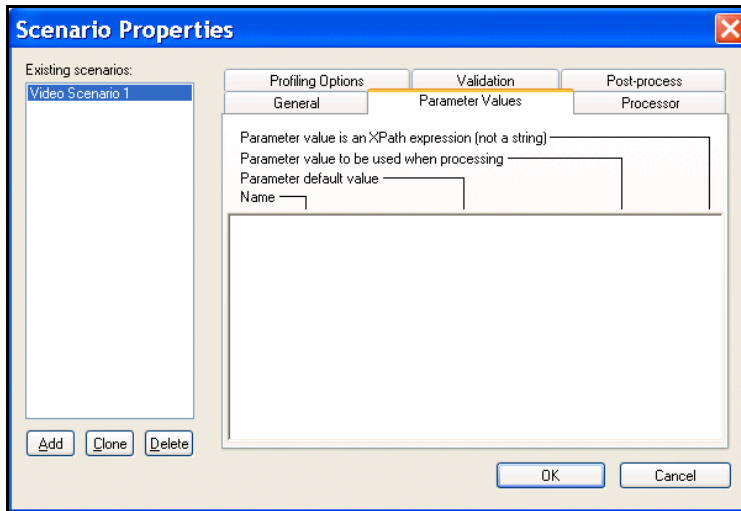


Figure 202. XSLT Scenario Parameter Values Tab

4. Click the third field in the line that displays the parameter for which you want to define a value – **Parameter value to be used when processing**.
5. Type the value of the parameter.
6. If you want Stylus Studio to pass this parameter as an XPath expression instead of as a string, click the fourth field, which is a check box.
The default is that Stylus Studio passes a parameter as a string.
7. Click **OK**.

- ◆ **To view stylesheet parameters and and specify stylesheet options, click the Params/Other tab in the stylesheet window.**

In the **XSLT Encoding** field, you can specify the encoding you want Stylus Studio to use when you save the stylesheet.

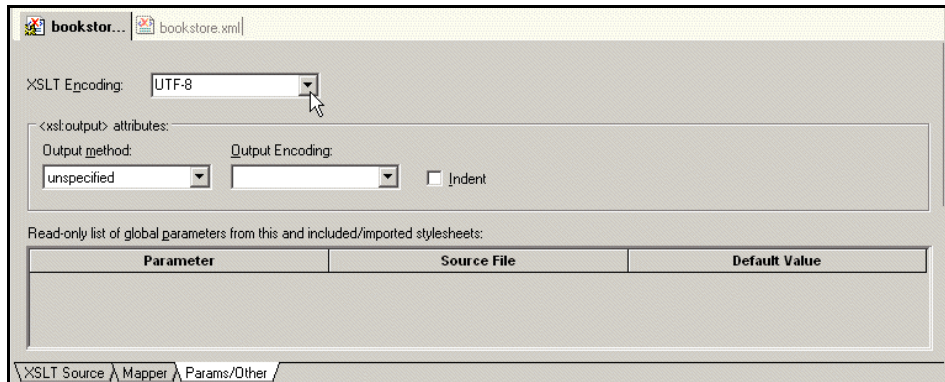


Figure 203. XSLT Parameters Tab

To display a list of the encodings supported by Stylus Studio, click the down arrow in the **XSLT Encoding** field.

In the **Output method** field, you can specify the type of data you want the stylesheet to generate. Choices include

- xml
- html — Stylus Studio generates HTML that is compliant with HTML 4.0. This is equivalent to inserting `<xsl:output method="html"/>` in a stylesheet.
- text
- unspecified

If you do not specify an `xsl:output` instruction in your stylesheet, Stylus Studio uses the default output method you specify here. If you do specify an `xsl:output` instruction in your stylesheet, that instruction overrides the default you specify here.

When the result of applying a stylesheet is XHTML, specify `xml` as the **Output method**. Note, however, that Stylus Studio displays rendered HTML in the **Preview in Browser** window.

In the **Params/Other** tab, in the **Output Encoding** field, you can specify the encoding you want Stylus Studio to use in the document that is the result of applying the stylesheet. When you apply a stylesheet, Stylus Studio uses this encoding for the output document.

You can change the encoding by changing the setting in the **Params/Other** tab or in the initial processing instruction in the stylesheet. When you change the setting in one of these places, Stylus Studio automatically changes it in the other. They are always the same. In the **Output Encoding** field, click the down arrow to display a list of the supported encodings.


If you want Stylus Studio to insert indents in the result document, select **Indent**.

Applying Stylesheets

In order to apply a stylesheet to an XML document, the stylesheet must be associated with a scenario. See

- [“Creating a Scenario”](#) on page 369
- [“Cloning Scenarios”](#) on page 371
- [“Saving Scenario Meta-Information”](#) on page 371

If your stylesheet is associated with a scenario, there are two ways to apply it:

- Click **Preview Result**  , which appears in the top tool bar of the **XSLT Source** tab of your stylesheet. This ignores any breakpoints that are set.
- Press F5. Stylus Studio suspends processing if it reaches a breakpoint.

The following topics provide more information about how to apply stylesheets:

- [About Applying Stylesheets](#) on page 366
- [Results of Applying a Stylesheet](#) on page 367
- [Applying Stylesheets to Large Data Sets](#) on page 369



Tip Stylus Studio provides a number of tools that help you debug stylesheets. See [“Debugging Stylesheets”](#) on page 493.

About Applying Stylesheets

When you apply a stylesheet, Stylus Studio checks both the XML source document and the XSLT stylesheet for correct syntax. If it detects any errors, it displays a message that indicates what the error is. This message appears at the bottom of the XSLT editor. Stylus Studio also displays and flags the line that contains the error.

Often, a stylesheet refers to other files, such as CSS stylesheets or images. For Stylus Studio to display the complete result file in the **Preview** window, you must enter the path for resolving any links. Do this in the **Base URL for HTML links resolution** field of the **Scenario Properties** dialog box.

Ensure that the correct output type is set. To do this, click the **Params/Other** tab at the bottom of the stylesheet and check the value of the **Output Method** field.



If your stylesheet is associated with more than one scenario, select the scenario you want to use and then apply the stylesheet. To do this, click the down arrow to the right of the scenario name field to display a list of scenarios. Click the scenario you want to use, and then click **Preview Result** . After you apply a stylesheet in a particular scenario, the **Preview** window displays a tab for that scenario. To reapply the stylesheet in that scenario, click **Preview Result**  in the left tool bar.

You might want to apply the same stylesheet to two different XML documents and compare the results. To do this, create a scenario for each XML source document. Apply the stylesheet in the context of each scenario. Stylus Studio displays a tab for each scenario at the bottom of the **Preview** window. Click the tab to display the result document for that scenario.

Stylus Studio does not support scenarios that consecutively apply multiple stylesheets to one source document. However, you can use the `StylusXslt` command-line utility within a batch file perform this type of operation. See [“Applying a Stylesheet from the Command Line”](#) on page 128.

Results of Applying a Stylesheet

Stylus Studio applies the stylesheet to the XML source document specified in the current scenario and refreshes the **Preview** window with the latest result document. If the **Preview** window is not visible, select **View > Preview** from the Stylus Studio menu bar.

To toggle between viewing the text of the result and viewing what the result would look like in a browser, click **Preview in Browser**  or **Preview Text** .

Tip You can select and copy text in the **Preview Text** view.

If you click in the result document, Stylus Studio displays the **Backmap Stack** window, which lists the XSLT instructions that generated the text you clicked.

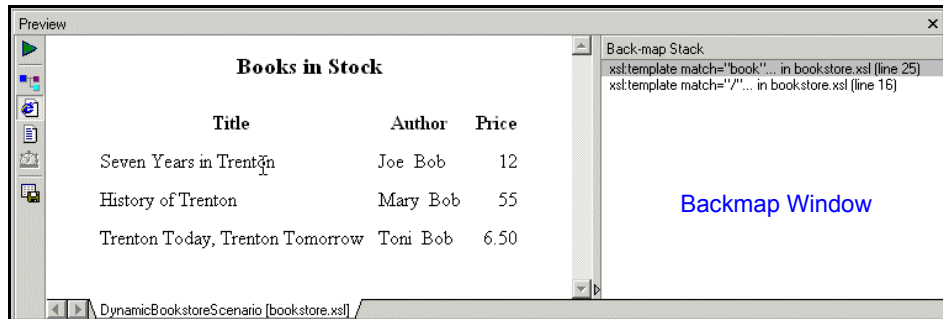



Figure 204. XSLT Backmap Window

Stylus Studio also flags the line in the stylesheet that contains the first instruction in the **Backmap Stack** window.

If the result document is XML, in the **Preview** window, you can click **Preview in Tree**  to display the result of XSLT processing as an XML tree.

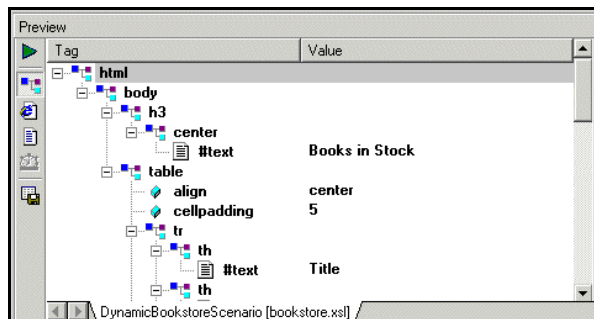




Figure 205. XML Tree View

The tree view provides

- Scalability – you can more easily view large result sets
- Backmapping – click on a result tree node and Stylus Studio displays the stylesheet line that generated that node

To save the result of applying a stylesheet, click **Save Preview**  in the **Preview** window. Stylus Studio displays the **Save As** dialog box.

The result document reflects any changes you made to either the XML source document or the XSLT stylesheet. You do not need to explicitly save either the XML or XSLT file to have changes to those documents appear in the result document. However, when you apply a stylesheet, Stylus Studio does not also save the stylesheet. To save a stylesheet, click **Save** .

Applying Stylesheets to Large Data Sets

When you open a stylesheet or assign a source XML document to a scenario, Stylus Studio loads the entire XML source document in memory. Stylus Studio requires the source XML document in order to display

- A preview of the result of applying the stylesheet
- The source tree for the document the stylesheet will be applied to

If the source XML document is particularly large, loading it can take several minutes.

Each time you leave and return to Stylus Studio, Stylus Studio checks whether any open documents have been modified. If the documents reside on remote servers, this can take some time. If you want, you can turn off the check for modified documents.

◆ To turn off the check for modified documents:

1. In the Stylus Studio menu bar, select **Tools > Options**.
Stylus Studio displays the **Options** dialog box.
2. Click **Application Settings**.
3. Click the check box for **Automatically check for externally modified files**.



Creating a Scenario

A scenario allows you to preview the results of applying a stylesheet. Each scenario is for a particular group of settings. These settings include the name of an XML source document, the values of any parameters in the stylesheet, and the values of any encoding settings. A scenario can include any setting that you can specify when you apply the stylesheet.

A scenario can be associated with only one stylesheet and only one XML source document. However, you can associate any number of scenarios with a stylesheet, and you can associate any number of scenarios with an XML source document.

Tip If you start to create a scenario and then change your mind, click **Delete** and then **OK**.

◆ **To create a scenario:**

1. In the XSLT editor tool bar, click **Browse**  .
Stylus Studio displays the **Scenario Properties** dialog box.
2. In the **Scenario Name:** field, type the name of the new scenario.
3. In the **Source XML URL:** field, type the name of the XML file you want to apply the stylesheet to, or click **Browse**  to navigate to an XML file and select it.
4. In the **Output URL** field, optionally type or select the name of the result document you want the stylesheet to generate. If you specify the name of a file that does not exist, Stylus Studio creates it when you apply the stylesheet.
5. In the **Base URL For HTML Links Resolution** field, optionally type the path for resolving any links. For example, your stylesheet might have links to CSS stylesheets or images.
6. If you want Stylus Studio to **Store paths relative to XSLT document path**, ensure that this option is checked.
7. If you want to **Preview result in an external application**, ensure that this option is checked. When this option is checked, Stylus Studio displays the result in the default application for the output method specified for the scenario. For example, if the output method for the scenario is HTML and if Internet Explorer is the default application for displaying HTML files, Stylus Studio displays the resulting HTML in Internet Explorer, as well as in the **XSLT Preview** window.
8. If you want to specify values for stylesheet parameters, click the **Parameter Values** tab. Double-click the **Value** field for the parameter you want to specify a value for.
9. If you want to use an XSLT processor other than the Stylus Studio processor, click the **Processor** tab and type the required information. See [“Using Third-Party XSLT Processors”](#) on page 387.
10. If you want to specify any post-processing, click the **Post-process** tab. See [“Post-processing Result Documents”](#) on page 393.
11. To define another scenario, click **Add** and enter the information for that scenario. You can also copy scenarios. See [“Cloning Scenarios”](#) on page 371.
12. Click **OK**.


For more information, see [“Scenario Properties General Tab \(XSLT\)”](#) on page 1199.

Cloning Scenarios

When you clone a scenario, Stylus Studio creates a copy of the scenario except for the scenario name. This allows you to make changes to one scenario and then run both to compare the results.

Tip If you start to clone a scenario and then change your mind, click **Delete** and then **OK**.

◆ To clone a scenario:

1. Display the stylesheet in the scenario you want to clone.
2. In the XSLT editor tool bar, click **Browse**  to display the **Scenario Properties** dialog box.
3. In the **Scenario Properties** dialog box, in the **Existing preview scenarios** field, click the name of the scenario you want to clone.
4. Click **Clone**.
5. In the **Scenario name** field, type the name of the new scenario.
6. Change any other scenario properties you want to change. See [“Creating a Scenario”](#) on page 369.
7. Click **OK**.

Saving Scenario Meta-Information

Stylus Studio can store scenario meta-information in two places:

- In the stylesheet associated with the scenario, unless you turned off the **Save scenario meta-information in the stylesheet** option. See [“Options - Module Settings - XSLT Editor - XSLT Settings”](#) on page 1164.
- In the project that the stylesheet belongs to, if the stylesheet belongs to a project.

When you save a stylesheet, Stylus Studio saves the scenario meta-information in the stylesheet, but not in the project. When you select **File > Save All** or when you save the project, Stylus Studio saves the scenario meta-information in the stylesheet *and* in the project. To ensure that scenario meta-information in the project and in the stylesheet is consistent

- The project must be open when you save the stylesheet.
- Ensure that you save the project after you modify scenario information. (If you close a project without saving it, Stylus Studio prompts you to save it.)

Suppose you modify a scenario and save and close the associated stylesheet. If the stylesheet belongs to the open project, when you save the project, Stylus Studio saves the closed stylesheet's scenario meta-information in the project.

Applying a Stylesheet to Multiple Documents

You can apply the same stylesheet to multiple documents


- [In separate operations](#)
- [In a single operation](#)

Applying the Same Stylesheet in Separate Operations

Scenarios make it easy to view results and apply the same stylesheet to multiple XML documents. A stylesheet can have any number of scenarios. Each scenario is associated with only one stylesheet. In addition to the stylesheet, a scenario is associated with a source XML file. The same XML file can be associated with any number of scenarios.

You create an initial scenario when you create a stylesheet. You can create additional scenarios at any time. See [“Creating a Scenario”](#) on page 369.

◆ **To view results for a particular scenario:**

1. Click the down arrow in the scenario field at the top of the stylesheet window.
2. Click the scenario you want to view.
3. Click **Preview Result**  , which is directly to the left of the scenario field. This applies the stylesheet to the XML document specified in the selected scenario.

Each time you generate a different scenario, Stylus Studio displays a tab at the bottom of the **Preview** window for that scenario. Click the tab for the scenario you want to view. This allows you to compare results.

Applying a Stylesheet to Multiple Documents in One Operation

To apply a stylesheet to multiple documents in one operation, call the `document()` function in the XPath expression of a template. This function allows you to access another XML document and select nodes from that document for processing as source nodes. See [“Accessing Other Documents During Query Execution”](#) on page 705.

For example, you can specify the following:

```
<xsl:apply-templates select="document('bookstore.xml')/bookstore"
```

This selects the bookstore root element of the bookstore.xml document.

Stylus Studio looks for the document in the directory that contains the stylesheet.

The document() function has a lot of overhead. You should call it once and assign the result to a variable with the `xsl:variable` instruction.

About Stylesheet Contents

Stylesheets are XML documents. They can contain XSLT instructions and non-XSLT elements and nodes. Stylus Studio automatically inserts some XSLT instructions. You can add additional XSLT instructions, HTML markup, and any other XML data you want.

This section describes

- [Contents Provided by Stylus Studio](#) on page 373
- [Contents You Can Add](#) on page 373

Contents Provided by Stylus Studio

When Stylus Studio creates a stylesheet, it has the following contents:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">

    </xsl:template>
</xsl:stylesheet>
```

The `xsl:stylesheet` instruction is required in every stylesheet that you use with Stylus Studio.

Stylus Studio defines one template, which matches the root node. Of course, the two built-in templates are also defined, although they are not explicitly in the stylesheet. For information about these templates, see [“Using Stylus Studio Default Templates”](#) on page 384.



When Stylus Studio creates a stylesheet from an HTML file, the template that matches the root node contains all HTML markup that was in the imported file.

Contents You Can Add


You can add to the stylesheet any XSLT instruction that Stylus Studio supports. See [“XSLT Instructions Quick Reference”](#) on page 408. You can also add HTML markup and any other XML-formatted data you require.

To obtain the XPath expression that retrieves a particular node in the source document you want to apply the stylesheet to, see [“Obtaining the XPath for a Node”](#) on page 161.

Updating Stylesheets

You can edit a stylesheet in the **XSLT Source** tab in **Full Source**  mode or **Template**  mode. To display a particular template in either mode, click the down arrow in the upper right corner of the editing pane. This displays a drop-down list of template match patterns. Click the template you want to view.

The XSLT editor keeps track of your XSLT context. That is, it keeps track of template match patterns, and any `xsl:for-each` element that affects the context on which the stylesheet is working. The editor uses Stylus Studio’s Sense:X technology to help you create XPath expressions whenever they are needed.

After you associate the stylesheet with a scenario, you can display the source tree for the XML source document specified in the scenario. Click **Source Tree**  in the XSLT editor tool bar. This tree provides a description of the structure of the XML source document specified in the scenario. This tree does not include elements and attributes that are not instantiated in the particular source document. However, the tree provides a structure that you can examine to help you understand stylesheet behavior in a given scenario.

The following sections describe the Stylus Studio editing tools:

- [Dragging and Dropping from Schema Tree into XSLT Editor](#) on page 374
- [Using Sense:X Automatic Tag Completion](#) on page 375
- [Using Sense:X to Ensure Well-Formed XML](#) on page 375
- [Using Standard Editing Tools](#) on page 376

Dragging and Dropping from Schema Tree into XSLT Editor

From the source tree of the XSLT editor, you can drag an element or attribute into the **XSLT Source** pane. If you drop the node in the stylesheet so that it is in a template, Stylus Studio displays the following choices:

- **xsl:for-each**
- **xsl:value-of**
- **xsl:apply-templates**
- *node_name*

Click the instruction you want to create. The XSLT context into which you drop the node determines the value of the `select` attribute in the instruction you choose. The `select` attribute always selects the node you dragged into the stylesheet. If you choose `node_name`, Stylus Studio simply inserts the name of the element or attribute you dragged in. This is convenient for pasting long element or attribute names.

If you drop the node in the stylesheet so that it is not in a template, Stylus Studio creates a new template. In the new template, the value of the `match` attribute is the name of the node you dragged into the stylesheet.

You can also create a new template by double-clicking a node in the source tree. The difference between double-clicking a node and dragging a node is that when you double-click a node to create a template, Stylus Studio always inserts the template at the end of the stylesheet. When you drag a node to create a template, you determine the location of the template.

Using Sense:X Automatic Tag Completion

The Stylus Studio Sense:X automatic tag completion system helps you edit XSLT, HTML, and FO (formatting objects) instructions. Stylus Studio has built-in knowledge of all XSLT, HTML, and FO tags, as well as their attributes.

Tip For information about FO, see <http://www.w3.org/TR/2001/REC-xs1-20011015/slice6.html#fo-section>.

As you type in the XSLT edit window, Stylus Studio prompts you with a list of tag or attribute names that match the first few letters you typed. To complete the tag name you are typing, scroll the list if necessary, and double-click the tag you want.

You can customize the Sense:X system. Edit `languages.xml` in the Stylus Studio `bin\Plugins\Configuration Files` directory to customize the tag list.

To set options that specify Sense:X behavior, see “[Options - General - Editor General](#)” on page 1142.

Using Sense:X to Ensure Well-Formed XML

Sense:X also helps you write well-formed XML. There is an option in the **Editor General** page that is set by default. This is **Auto-Close Open Tag When Typing '</'**. This means that as soon as you type `</`, Stylus Studio immediately inserts the only tag that can possibly be closed at that point.

If you prefer, you can turn off this option. Then, when you start to type a closing tag, the Sense:X list displays the only valid closing tag. Double-click it to insert it.

Using Standard Editing Tools

Standard editing tools are available to you for updating stylesheets. From the **Edit** menu or tool bar you can cut, copy, paste, replace, undo, redo, select all, and find. The usual keyboard shortcuts work as well:


- Ctrl+X cuts highlighted text.
- Ctrl+C copies highlighted text.
- Ctrl+V pastes text.
- Ctrl+Z undoes the most recent action that has not already been undone.
- Ctrl+Y redoes the most recently undone action that has not already been redone.

For additional shortcuts, see “[Keyboard Accelerators](#)” on page 1116.

Saving Stylesheets

When you save a stylesheet, Stylus Studio uses the encoding that is specified in the **Params/Other** tab of the XSLT editor. You can change the encoding by changing the setting in the **Params/Other** tab or in the initial processing instruction in the stylesheet. When you change one of these, Stylus Studio automatically changes the other. They are always the same.

To save an XSLT stylesheet, do one of the following:

- Click **Save**  .
- Press Ctrl+S.
- Select **File > Save** from the Stylus Studio menu bar.

To save your stylesheet to another file, select **File > Save As**.

To save multiple files, select **File > Save All**. This saves all files that are open in Stylus Studio.

Tip You can set an option that instructs Stylus Studio to save your modified documents every few minutes. See “[Options - Application Settings](#)” on page 1129.

Using Updated Stylesheets

Within a scenario, Stylus Studio automatically uses any updated files when you apply a stylesheet. It does not matter whether you have explicitly saved a file in the scenario. If a stylesheet includes or imports other stylesheets, Stylus Studio automatically uses any updated versions of included or imported stylesheets even if you have not explicitly saved them.

However, there is one situation in which Stylus Studio does not automatically use updated stylesheets. Suppose that multiple stylesheets are open in Stylus Studio. Each stylesheet generates a Web page, and the Web pages have links to each other. The stylesheets do not include or import each other. You make changes in more than one of these stylesheets and you do not explicitly save any changes. You apply one of the stylesheets, and in the **Preview** window you click a link to another Web page generated by one of the other stylesheets you updated. In this situation, Stylus Studio does not apply the updated stylesheet. You must explicitly save the stylesheet to be able to use the updated version.

Specifying Extension Functions in Stylesheets

You can write XSLT extension functions in Java and invoke them in XPath expressions in stylesheets. This section provides instructions for implementing and invoking extension functions from your stylesheet.

This section covers the following topics:

- [Using an Extension Function in Stylus Studio](#) on page 378
- [Basic Data Types](#) on page 379
- [Declaring an XSLT Extension Function](#) on page 379
- [Working with XPath Data Types](#) on page 380
- [Declaring an Extension Function Namespace](#) on page 380
- [Invoking Extension Functions](#) on page 381
- [Finding Classes and Finding Java](#) on page 381
- [Debugging Stylesheets That Contain Extension Functions](#) on page 381

Using an Extension Function in Stylus Studio

The process of using an extension function in Stylus Studio involves three main steps:

1. First, you need to write a Java class that can be used from within a stylesheet. In this example, the `SystemDate()` method returns the system date and time as a string:

```
import java.util.Date;
public class SystemUtils
{
    public Object SystemDate()
    {
        Date d = new Date();
        String s = d.toString();
        return s;
    }
}
```

2. Second, compile your class and register it on the Stylus Studio host by copying the `.class` file to a location defined in the host's `CLASSPATH` environment variable.
3. Finally, specify information in the stylesheet so that Stylus Studio can use your class. You do this with a namespace reference in the `xmlns:stylesheet` tag. For example, define a namespace as `xmlns:Ext` where `Ext` is the prefix to use when calling the class methods. (`Ext` is not a predefined keyword; it can be replaced by any other legal string.) The namespace reference then takes the class name as a value. In this example, the whole reference looks like the following:

```
xmlns:Ext="SystemUtils"
```

The class is now available from within the stylesheet and can be used in a template such as the following:

```
<xsl:template match="NODE">
  <p><xsl:value-of select="Ext:SystemDate()"/></p>
</xsl:template>
```


The XSLT stylesheet might look like the following:

```
<?xml version="1.0" encoding="ISO-10646-UCS-2"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform"
xmlns:Ext="SystemUtils">
  <xsl:param name="param">test</xsl:param>
  <xsl:template match="*/">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="text()|@">
    <xsl:value-of select="."/>
  </xsl:template>
  <xsl:template match="NODE">
    <p><xsl:value-of select="Ext:SystemDate()"/></p>
  </xsl:template>
</xsl:stylesheet>
```

Basic Data Types

XPath and XSLT data types map to Java data types according to [Table 51](#):

Table 51. XPath/XSLT and Java Type Mappings

<i>XPath/XSLT Type</i>	<i>Java Type</i>
Node Set	org.w3c.dom.NodeList
String	java.lang.String
Boolean	boolean or Boolean
Number	double or Double
Result Tree Fragment	org.w3c.dom.DocumentFragment

Declaring an XSLT Extension Function

Extension functions must have one of the following signatures:

```
public Object FxnName()
public Object FxnName(Type1 var1, Type2 var2,...)
public static Object FxnName()
public static Object FxnName(Type1 var1, Type2 var2,...)
```

A class that contains an extension function might look like the following:

```
import org.w3c.dom.*;
import java.lang.Double;
public class NumberUtils
{
    public Object Average(NodeList nl)
    {
        double nSum = 0;
        for (int i = nl.getLength() - 1; i >= 0; i--)
        {
            nSum +=
                Double.valueOf(nl.item(i).
                    getNodeValue()).doubleValue();
        }
        return new Double(nSum / nl.getLength());
    }
}
```

Working with XPath Data Types

The XPath types `Boolean` and `Number` can map either to the corresponding Java primitive types or to the corresponding Java object types. If the XPath processor is looking for a function that accepts XPath parameters 3.2 and `true`, it looks first for a function that accepts `(double, boolean)` and then `(Double, Boolean)`. Functions that accept some combination of primitive types and object types are not recognized by the XPath processor.

The XPath processor determines the actual return type of a function at run time. For example, the XPath processor treats the return type of the function in the preceding section as an XPath `Number` because the object it returns is an instance of the Java class `Double`. You must declare all functions to return type `Object`, regardless of the actual type of the return value.

Declaring an Extension Function Namespace

In conformance with the XSLT specification, extension functions are accessed through a unique namespace. The namespace declaration can be in any of the following locations:

- `xmlns:stylesheet` tag
- Element that contains the XPath expression that invokes the extension function
- Ancestor of the element that contains the XPath expression that invokes the extension function

The XPath processor treats the namespace URI as a fully qualified class name. If the class name is preceded by `class:`, all calls are to static methods only. Otherwise, an instance of

the class is created on first use and released when stylesheet processing is complete. Performance is better when you use a static method because creation and deletion of an instance of the class is not required.

You can separate package names with either a dot (.) or a forward slash (/). An sample namespace declaration might look like the following:

```
<xsl:stylesheet xmlns:Ext="NumberUtils">
```

The XPath processor resolves namespace prefixes in names of extension functions relative to the namespace declarations in the stylesheet.

Invoking Extension Functions

You use XSLT extension functions just like built-in XPath functions. For example:

```
<xsl:value-of select="Ext:Average(portfolio/stocks/last)"/>
```

Finding Classes and Finding Java

The XPath processor looks for extension classes by using the CLASSPATH environment variable. Ensure that your CLASSPATH references any directories or .jar files that contain extension classes.

The XPath processor tries to load the Sun Java Runtime Environment (JRE) 1.4.x or later. If the XPath processor cannot find a suitable JRE, invoking Java extension functions causes an error during stylesheet processing.

In Stylus Studio, classes are reloaded each time you refresh the preview output, so changes in a class are reflected in subsequent preview output.

Debugging Stylesheets That Contain Extension Functions



Support for extensions debugging is available only in Stylus Studio XML Enterprise Suite and Stylus Studio XML Professional Suite.

You can use Stylus Studio backmapping and debugging features on stylesheets that invoke extension functions. You must process the stylesheet with one of the following processors:

- Built-in Stylus Studio XSLT processor
- Saxon processor

The Saxon processor does not allow you to step into JavaScript extensions. You can step into Java extensions, however.

Working with Templates

Templates define the actions that you want the XSLT processor to perform. When you apply a stylesheet to an XML source document, the XSLT processor populates the result document by instantiating a sequence of templates. This is illustrated in [“Understanding How the Default Templates Work”](#) on page 353.

A template can contain elements that specify literal result nodes. It can also contain elements that are XSLT instructions for creating result nodes. In a template, the template rule is the pattern that the XSLT processor matches against (compares with) selected nodes in the source document.

This section covers the following topics:

- [Viewing Templates](#) on page 382
- [Using Stylus Studio Default Templates](#) on page 384
- [Creating Templates](#) on page 386
- [Applying Templates](#) on page 387
- [Updating Templates](#) on page 387
- [Deleting Templates](#) on page 387

Viewing Templates

Stylus Studio provides different ways to display lists of templates, specific templates, as well as ways to see if a given template generates any output.

Viewing a List of Templates

◆ To view a list of the templates in the stylesheet:

1. Click the down arrow in the upper right corner of the XSLT editing pane.

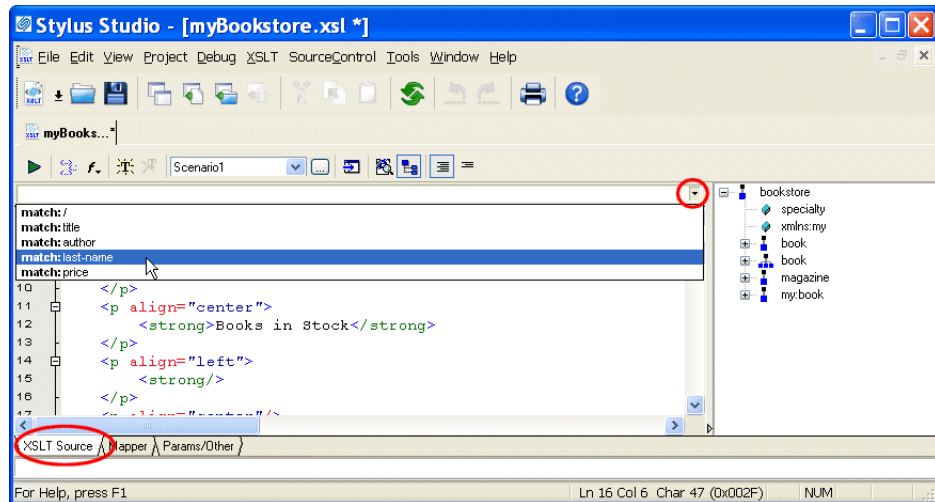




Figure 206. Choosing Available XSLT Templates

Stylus Studio displays a drop-down list of the first five match patterns in the stylesheet. To limit the displayed list, type in the combo box to the left of the down arrow. Stylus Studio displays only those patterns that match the character(s) you typed.

Note A particular template might or might not have a match pattern (template rule). Named templates do not necessarily specify match patterns.


2. Click the match pattern for the template you want to view. It does not matter whether the XSLT editor is in **Full Source**  mode or **Template**  mode.

Viewing a Specific Template

To view a particular template, double-click its matching element in the XML tree view, which is displayed to the right of the editing pane. If the element has more than one template, Stylus Studio displays a list of the templates. Click the one you want.

Checking if a Template Generates Output

◆ **To see if a particular template generates any output:**

1. Select a template in the XSLT templates pane.
2. Click **Refresh**  to apply the stylesheet.
3. In the **XSLT Preview** window, with output text displayed, scroll as necessary to find text highlighted in gray. Text with a gray background was generated by the selected template.

Using Stylus Studio Default Templates

Every stylesheet in Stylus Studio can use two built-in templates, even though they are not explicitly defined. This section covers the following topics to help you use these templates:

- [Contents of a New Stylesheet Created by Stylus Studio](#) on page 384
- [About the Root/Element Built-In Template](#) on page 385
- [About the Text/Attribute Built-In Template](#) on page 385

Contents of a New Stylesheet Created by Stylus Studio

When Stylus Studio creates a new stylesheet, the stylesheet includes the following built-in templates:

```
<xsl:template match="*/">  
  <xsl:apply-templates/>  
</xsl:template>  
<xsl:template match="text()|@">  
  <xsl:value-of select="."/>  
</xsl:template>
```

Every XSLT stylesheet contains these templates whether or not they are explicitly specified. In other words, the XSLT processor behaves as if they are there even when they are not explicitly specified in the stylesheet.

About the Root/Element Built-In Template

The first built-in template matches `*|/`. This means it matches every element in the source document and it matches the root node. This is the root/element built-in template.

This root/element built-in template contains only the `xs1:apply-templates` instruction. The `xs1:apply-templates` instruction does not specify a `select` attribute, which means that the XSLT processor operates on the children of the node for which the root/element template was instantiated.

What does the XSLT processor do when it operates on these children nodes? It searches for a template that matches each node. If there is no such template and if the node is an element, the XSLT processor instantiates the root/element built-in template. If the node is a text node and there is no matching template, the XSLT processor instantiates the text/attribute built-in template.

If the node for which the root/element built-in template is instantiated has no children, the XSLT processor does no processing for this node and proceeds to the next selected node.

The XSLT processor instantiates the root/element built-in template when it cannot find a template that explicitly matches the root node or an element in the source document. As you know, the XSLT processor always begins processing by instantiating the template that matches the root node. If you do not define such a template in your stylesheet, the XSLT processor begins processing by instantiating the root/element built-in template.

About the Text/Attribute Built-In Template


The second specified built-in template matches `text()|@*`. This means it matches the text contents of every text node and every attribute in the source document. This is the text/attribute template.

This template contains only the `xs1:value-of` instruction. Its `select` attribute specifies an expression for selecting an XML node. The `"."` expression identifies the current node, which is the node the template was instantiated for.

This template copies the text contained in the current text node or attribute to the result document.

Creating Templates

To do anything beyond copying the text from your XML document to the result document, you must create templates. You can create new templates several ways:

- In the source tree of the XSLT editor, double-click the element or attribute for which you want to create a template. Stylus Studio creates an empty template that matches the node you clicked. This template appears at the end of the stylesheet.
- Click **New Template** . Stylus Studio creates an empty template whose match pattern is **NewTemplate**. Replace **NewTemplate** with a match pattern that has meaning for your stylesheet. The new template appears at the end of the stylesheet.
- Drag an element or node from the source tree in the XSLT editor to the **Full Source** pane and drop it in a location that is not in a template. Stylus Studio creates a new template that matches the node you dragged in.

◆ **Try creating a new template that matches an XML element in your document:**

1. Double-click an element in the tree view of your XML document.
2. Enter the following instruction in the new template:
`<xsl:value-of select="."/>`
3. In another template, ensure that there is an `xsl:apply-templates` instruction that selects the new template's element for processing.
4. Press F5 to apply the stylesheet and refresh the current scenario in the **Preview** window.

Notice that the text contents of the element for which you created the template are now displayed in bold – the XSL instruction is formatted with `` and ``. Also, the XSLT processor does not process this element's children (if there are any) because the new template you created does not specify `<xsl:apply-templates/>`.

By creating additional templates to style portions of your XML document, you can completely control how the document appears.

Saving a Template

To save a template, save the stylesheet. Click **Save**  in the Stylus Studio tool bar, or select **File > Save** from the Stylus Studio menu bar.

Applying Templates

The `xmlns:apply-templates` instruction allows you to control the order of operations when you apply a stylesheet. For an in-depth description of how the XSLT processor applies templates, see [“How the XSLT Processor Applies a Stylesheet”](#) on page 333.

To apply a template so that you can see the output in the **Preview** window, you must apply the entire stylesheet. Press F5 to apply the stylesheet and refresh the output. If Stylus Studio detects any errors in the stylesheet or in the XML source document, it displays a message that indicates the cause and location of the error.

In the **Preview** window, in the **Text** view, the text with a gray background was generated by the template the cursor is in. If the editor is in **Template** mode, the text with the gray background was generated by the currently visible template.

Updating Templates

When you want to update a template, you can use all features that are available when you are updating a stylesheet. See [“Updating Stylesheets”](#) on page 374.

Deleting Templates

◆ **To delete a template:**

1. Select the text for the template you want to delete.
2. Right-click in the editor to display the shortcut menu.
3. Click **Cut**.

Using Third-Party XSLT Processors

In addition to a built-in XSLT processor, Stylus Studio includes several third-party XSLT processors, including Saxon (8.x and 6.x), MSXML, and .NET. Note, however, that only the following XSLT processors support Stylus Studio stylesheet debugging and back-mapping functionality:

- Stylus Studio’s built-in XSLT processor
- Saxon 8.x and 6.x
- Microsoft .NET (XslTransform and XslCompiledTransform)

Back-mapping and debugging are not supported by other XSLT processors, including others bundled with Stylus Studio (MSXML 4.0, for example).

Note You can use only the built-in XML parser with Stylus Studio.

This section covers the following topics:

- [How to Use a Third-Party Processor](#) on page 388
- [Setting Default Options for Processors](#) on page 390

How to Use a Third-Party Processor

You specify XSLT processors for stylesheets individually. You can, of course, create multiple scenarios for the same stylesheet, with each one using a different processor. When you use a third-party XSLT processor, output from the processor appears in the **Preview** window.

◆ To use a third-party XSLT processor:

1. Open the stylesheet.
2. In the XSLT Editor, in the scenario name field, click the down arrow to display the scenarios associated with the stylesheet.
3. To use a third-party processor for an existing scenario, click the scenario name, and then click to display the **Scenario Properties** dialog box.
To create a new scenario, click **Create Scenario**. See [“Creating a Scenario”](#) on page 369.
4. In the **Scenario Properties** dialog box that appears, click the **Processor** tab.
5. Select the XSLT processor you want to use from the **Processor** drop-down list.

Note Stylus Studio allow you to choose an MSXML option only if you have the MSXML or MSXML .NET Framework (version 1.1 or 2.0) installed.

6. Optionally, change the default settings for the processor you selected.
7. If you selected a standard XSLT processor, you are done. Click **OK**.
If you selected **Use custom processor (%1 xml, %2 xslt, %3 output)**:
 - a. In the **Command line** field, type the command line for invoking the processor you want to run. You must specify the command line so that it is clear where to use the three arguments. Following are two examples:

```
myparser %1 %2 %3
```

```
myparser -inputxslt %2 -inputxml %1 -out %3
```

- b. In the **Path** field, specify any path that needs to be defined for the processor to run. Typically, this is the location of the processor.
- c. In the **Classpath** field, type any directories the external processor needs to access that are not already specified in your CLASSPATH environment variable.
- d. Click **OK**.

Using the Saxon Processor

Stylus Studio lets you execute XSLT documents using either the Saxon-B (basic) or Saxon-SA (schema-aware) processor. You specify which processor you want to use with the **Execution mode** property in the **Saxon XSLT Settings** dialog box. Settings that have command line equivalents in Saxon show the command in parentheses following the property name. Some settings are available only if you are using Saxon-SA.

Stylus Studio's Sense:X syntax coloring and auto-completion provides full support for Saxon syntax, so long as the Saxon Logic XSLT processor is either associated with the current XSLT scenario or has been set as the default XSLT processor.

If you want to use the Saxon processor:

1. On the **Processors** tab, click **Saxon**.
The **Settings** button becomes active.

2. Click the **Settings** button.

The **Saxon XSLT Settings** dialog box appears.

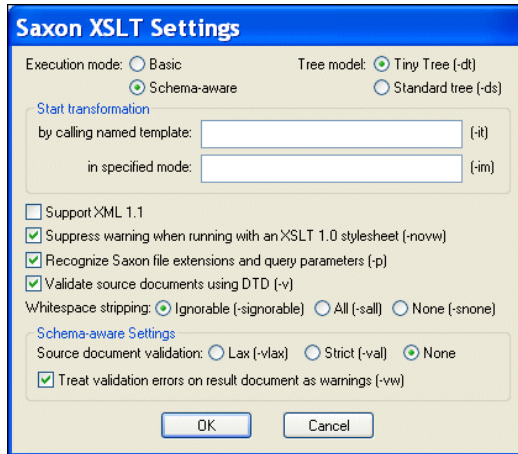


Figure 207. Saxon XSLT Settings Dialog Box

3. Complete the settings as desired. Press F1 to access the Stylus Studio online help, or refer to the [Saxon documentation](#) for more information.
4. Click **OK**.

Passing Parameters

To pass parameters to the Stylus Studio built-in XSLT processor or to the Microsoft .NET, Saxon, or MSXML processor, specify them in the **Parameters** tab of the **Scenario Properties** dialog box. To pass parameters to a custom external processor, you must specify them in the command line you enter.

Setting Default Options for Processors

If you want, you can set default values for XSLT processor options and designate a processor other than the built-in processor as the default processor used whenever you create an XSLT scenario.

You can always override the default processor and individual processor settings at the scenario level.

◆ **To set defaults for XSLT processors:**

1. From the Stylus Studio menu, select **Tools > Options**.
Stylus Studio displays the **Options** dialog box.
2. Select **Module Settings > XSLT Editor > Processor Settings**.

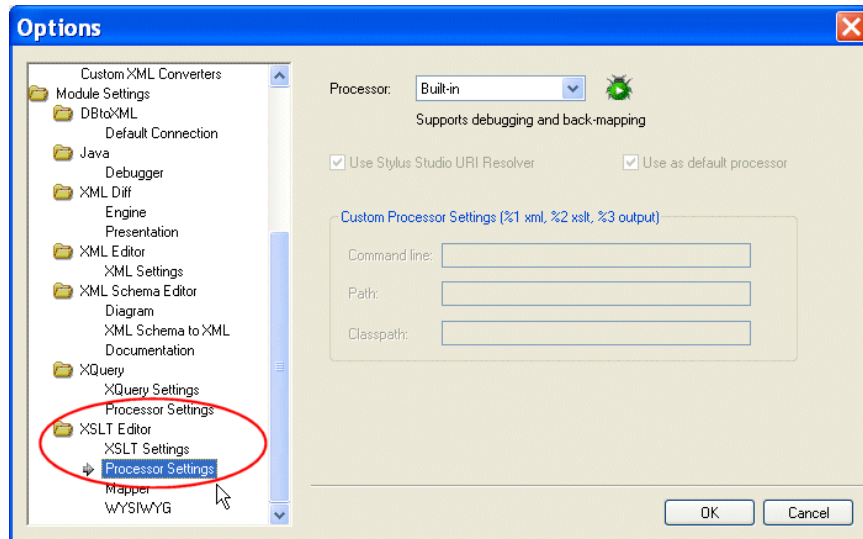


Figure 208. Options for XSLT Processors

3. Select the processor for which you want to specify default settings from the **Processor** drop-down list.
4. If required, complete processor-specific settings. (Click the **Settings** button.)
5. If you want this processor to be used as the default processor for all XSLT scenarios, click the **Use as default processor** check box.
6. Click **OK**.


Validating Result Documents

You can optionally validate the XML document that results from XSLT processing. You can validate using the

- Stylus Studio built-in processor (Xerces C++). If you use the Stylus Studio built-in processor, you can optionally specify one or more XML Schemas against which you want the result document to be validated.
- Any of the customizable processors supported by Stylus Studio, such as the .NET XML Parser and XSV.

All validation is done before any post-processing that you might have specified.

◆ **To validate XSLT scenario result documents:**

1. Open the stylesheet whose results you want to validate.
2. In the XSLT Editor (**XSLT Source** or **Mapper** tabs), in the scenario name field, click the down arrow and click the name of the scenario for which you want to perform validation.
3. Click **Browse**  to open the **Scenario Properties** dialog box.
4. Click the **Validation** tab.

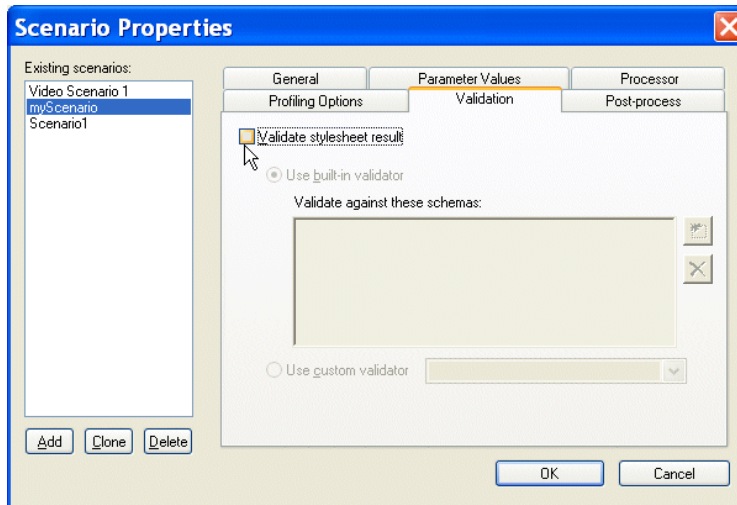



Figure 209. Validation Tab for XSLT Scenarios

5. Click **Validate stylesheet result**.


6. If you are using Stylus Studio's built-in validation engine, optionally, specify the XML Schemas against which you want to validate the XML result document. Otherwise, go to [Step 7](#)
 - a. Click the Open file button ().
The **Open** dialog box appears.
 - b. Select the XML Schema you want to use for validation.
 - c. Click the **Open** button to add the XML Schema to the **Validation** tab.
 - d. Optionally, add other XML Schemas.
 - e. Go to [Step 8](#).
7. Click the **Use custom validator** button, and select the validation engine you want to use from the drop-down list box.
8. Click **OK**.

Post-processing Result Documents

You can use a scenario's post-processing settings to specify that you want Stylus Studio to initiate processing on the result of applying a stylesheet. If you do, Stylus Studio performs the post-processing before it displays the result in the **Preview** window. Stylus Studio can postprocess the result of its built-in XSLT processor or an external XSLT processor.

You can choose to run the Apache Software Organization's Formatting Objects Processor (FOP) as a postprocessor. You can run this on the result of stylesheets that generate XML documents that contain FO. The Apache FOP included with Stylus Studio converts FO XML into PDF and displays it in the Stylus Studio preview window. See "[Generating Formatting Objects](#)" on page 394.

◆ To specify post-processing:

1. Open the stylesheet whose result you want to process.
2. In the XSLT Editor (**XSLT Source** or **Mapper** tabs), in the scenario name field, click the down arrow and click the name of the scenario in which you want to specify post-processing.
3. Click **Browse**  to open the **Scenario Properties** dialog box.
4. Click the **Post-process** tab.

5. Click one of the following:

Post Process With Apache FOP if you want Stylus Studio to initiate the Apache FOP. You are done. Click **OK**.

Custom Post-Process if you want Stylus Studio to initiate a postprocess you define. With this selection, you must also do the following:

- a. In the **Command line** field, type the command line for starting your postprocessor. For example, `mypostprocessor %1 %2`. You can specify any application or script that takes as input the result document generated by an XSLT processor and generates a new file.
- b. In the **Generated File Extension** field, type the extension on the file name of the postprocessor output. For example, `.pdf`.
- c. In the **Additional Path** field, optionally type any paths that need to be defined that are not already defined in your `PATH` environment variable.
- d. Click **OK**.

Generating Formatting Objects

You can use Stylus Studio to develop a stylesheet that generates XSL Formatting Objects (FO). In the scenario in which you apply such a stylesheet, you can specify that Stylus Studio should run a Formatting Objects Processor (FOP) on the stylesheet's result document. When you apply the stylesheet and preview the results, Stylus Studio displays the formatted results.

Stylus Studio includes The Apache Software Organization's FOP, and it is configured to always generate PDF. If you want to run a FOP to generate some other type of output, you must specify some other FOP in the **Custom post-process** fields of the **Post-process** tab of the **Scenario Properties** dialog box.

Stylus Studio includes two sample stylesheets that generate formatting objects. These files are in the `examples\XSLFormattingObjects` directory of your Stylus Studio installation directory.

This section covers the following topics:

- [Developing Stylesheets That Generate FO](#) on page 395
- [Troubleshooting FOP Errors](#) on page 395
- [Viewing the FO Sample Application](#) on page 396
- [Deploying Stylesheets That Generate FO](#) on page 398

- [Using Apache FOP to Generate NonPDF Output](#) on page 399

Note FO is a W3C recommendation for an XML vocabulary that describes how to format text. FO is one part of XSL. This section assumes that you are familiar with FO. For additional information about FO, see <http://www.w3.org/TR/2001/REC-xsl-20011015/>.

Developing Stylesheets That Generate FO

◆ To develop a stylesheet that generates FO:

1. Define the scenario in which you want to apply the stylesheet that generates FO. See [“Creating a Scenario”](#) on page 369.
2. In the **Scenario Properties** dialog box, in the **Post-process tab**, do one of the following:
 - Select **Postprocess with Apache FOP**.
The Apache FOP included with Stylus Studio is configured to convert FO XML into PDF. Stylus Studio then uses Acrobat Reader to display the PDF in the Stylus Studio preview window.
 - Specify some other FOP in the **Custom post-process** fields. You must do this when you want to generate output other than PDF. If you want to use the Apache FOP included with Stylus Studio to generate a format other than PDF, you can do that here.

See [“Post-processing Result Documents”](#) on page 393.


3. In the XSLT editor, define a stylesheet that generates FO. As soon as you type `<fo:`, Stylus Studio displays a completion menu of FO that you can select from.
4. Apply the stylesheet to an XML document.
After Stylus Studio transforms the XML document to generate a result XML document that contains formatting objects, Stylus Studio automatically runs the FOP you specified on the result document. Stylus Studio then displays the postprocess result in the **XSLT Preview** window.

Troubleshooting FOP Errors

If the transformation works well, but the FOP generates an error, obtain a copy of RenderX’s *Unofficial DTD for XSL Formatting Objects*. You can find this at <http://www.renderx.com>. Although this DTD is not official (it is more limited than what the W3C XSL recommendation defines), it is a helpful debugging tool.

◆ **To validate the generated XML against this DTD:**

1. Copy the DTD to a location such as C:\fo.dtd.
2. Include a document type declaration, such as the following, in your generated document:

```
<!DOCTYPE fo:root SYSTEM "/fo.dtd">
```
3. Turn off post-processing.
4. Apply the stylesheet.
5. Save the resulting XML document.
6. Open the saved XML document in Stylus Studio.
7. Click **Validate Document**  .

Viewing the FO Sample Application

◆ **To view the FO sample application included with Stylus Studio:**


1. In Stylus Studio, open the examples\XSLFormattingObjects\minimal-catalog.xml file in your Stylus Studio installation directory.

Alternative: If the Stylus Studio examples project is open, you can access this file from the **Project** window. To open the examples project, open examples.prj in the Stylus Studio examples directory.

The video scenario has already been defined. In the Post-process tab of the Scenario Properties dialog box, **Postprocess with Apache FOP** is selected.

In this scenario, Stylus Studio selects elements to operate on from three different documents. These documents are in the examples directory of the Stylus Studio installation directory. They are also in the examples project. The documents are:

- VideoCenter\videos.xml
- simpleMappings\books.xml
- simpleMappings\catalog.xml

2. Click **Preview Result**  . As you can see, the **Output Window** shows some post-processing information messages.

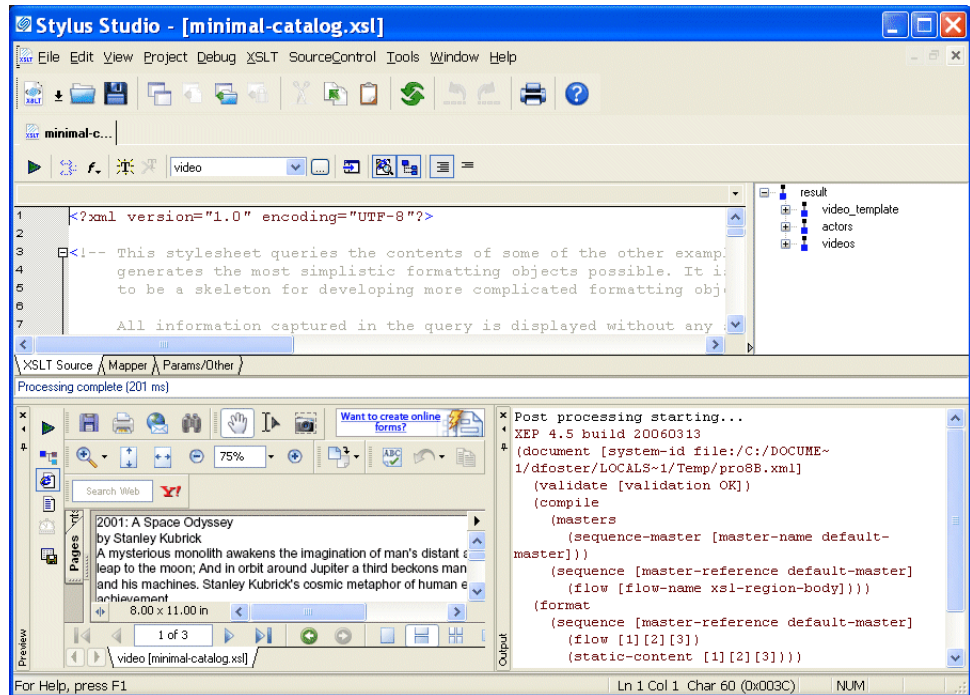


Figure 210. Example of XSLT FO Processing

After a few seconds, the **Preview** window displays the PDF result in Acrobat Reader. The result contains a few lines of text for each video and book found in the XML source documents. The title, author or director, and the description is included for each item. It is hard to see where information for one item ends and another begins.

3. Examine the stylesheet. It contains the minimum FO instructions required to generate FO. There is no formatting to make the result document easier to read. You can use this stylesheet as a skeleton for creating your own stylesheets that generate FO.
4. Now open the `examples\XSLFormattingObjects\catalog.xsl` stylesheet.

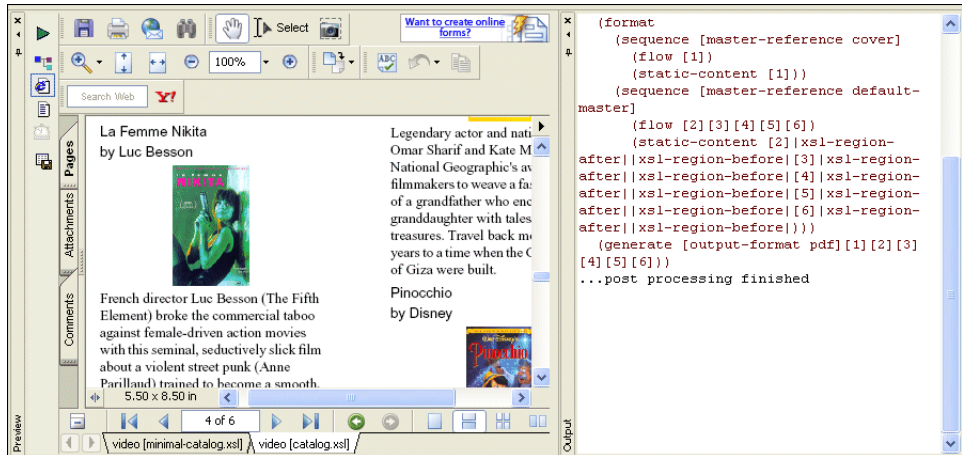
5. Click **Preview Result**  .

Figure 211. Another Example of XSLT FO Processing

This time the PDF result in the **Preview** window is nicely formatted. The `catalog.xml` stylesheet adds some basic formatting, as well as images, to the `minimal-catalog.xml` stylesheet. Now it is easy to distinguish the title, author or director, and description for each video or book.

Deploying Stylesheets That Generate FO

When your stylesheet is complete, the process for creating a final document, such as a PDF document, from an XML document is as follows:

1. Apply a stylesheet to an XML document. This results in an XML document that contains XSL FO.
2. Run a FOP, such as Apache's FOP, and use the generated XML as input.

Example

You can accomplish both steps with a single invocation of FOP on the command line. For example:

```
java -cp "C:\Program Files\StylusStudio\bin\Plugins\Fop\fop.jar;"
org.apache.fop.apps.Fop -xml ..\VideoCenter\videos.xml -xsl
catalog.xml -pdf multimediacatalog.pdf
```

Replace C:\Program Files\StylusStudio with the name of the directory in which Stylus Studio is installed.

Using Apache FOP to Generate NonPDF Output

The Apache FOP included with Stylus Studio is configured to output PDF.

◆ **To use this FOP to generate some other type of output:**

1. Open the stylesheet whose results you want to postprocess with the Apache FOP.
2. Create or open a scenario in which to do the post-processing. See [“Creating a Scenario”](#) on page 369. Stylus Studio displays the **Scenario Properties** dialog box.
3. In the **Scenario Properties** dialog box, click the **Post-process** tab.
4. Select **Custom post-process**.
5. In the **Command line** field, enter something like the following:

```
java -cp "C:\Program Files\StylusStudio\bin\Plugins\Fop\fop.jar"
org.apache.fop.apps.Fop -fo %1 -svg %2
```

Modify this sample command line according to where Stylus Studio is installed and what kind of output you want the FOP to generate. The last option, -svg in the example, can be any of the following:

Table 52. FOP Output Options

Setting	Output
-mif	MIF file
-pcl	PCL file
-txt	Text file
-svg	SVG slides file
-at	XML (representation of an area tree)
-pdf	PDF file

6. In the **Generated file extension** field, specify the extension that indicates the type of output you want. For example, specify .txt if you want the FOP to generate a text file.

7. If there is additional path information that the FOP will require to execute successfully, type it in the **Additional path** field.
8. Click OK.

Generating Scalable Vector Graphics

The procedure for defining a stylesheet that generates an XML document that contains Scalable Vector Graphics (SVG) is the same as for any other stylesheet. Simply create a stylesheet that specifically creates SVG elements. You can then use Stylus Studio to display the rendered SVG.

Note SVG is a W3C recommendation for an XML vocabulary that describes two-dimensional graphics. It is assumed that you are familiar with SVG. For additional information about SVG, see <http://www.w3.org/Graphics/SVG>.

About SVG Viewers

If you have an installed SVG viewer, Stylus Studio automatically displays the rendered SVG when you apply the stylesheet.

If you do not have an installed SVG viewer, you can still define a stylesheet that generates SVG. However, when you try to preview the result of the stylesheet, Stylus Studio displays the generated XML. You can download an SVG viewer from Adobe Systems Incorporated at <http://www.adobe.com/support/downloads/main.html>. Under **Readers**, select the SVG Viewer for **Windows**. After you install an SVG viewer, you must restart Stylus Studio and Internet Explorer to be able to view the rendered graphics.


Running the SVG Example

◆ **To run the SVG example that is included in Stylus Studio:**

1. In Stylus Studio, open the `examples\SVG\chart.xsl` file in your Stylus Studio installation directory.

Alternative: If the Stylus Studio `examples` project is open, you can access this file from the **Project** window. To open the `examples` project, open `examples.prj` in the Stylus Studio `examples` directory.

The SalesFigures scenario has already been defined. In this scenario, the stylesheet operates on elements in the `chart.xml` source document. This file is also in the `examples` project, and in the `examples\SVG` directory.

2. Click **Preview XSLT Result** . Stylus Studio automatically uses your installed SVG viewer to render the resulting XML and display the SVG.

If you do not have an SVG viewer installed, Stylus Studio displays the resulting XML.

Generating Java Code for XSLT



Java code generation is available only in Stylus Studio XML Enterprise Suite.

Stylus Studio includes a Java Code Generation wizard that creates Java code based on the scenarios defined for an XSLT document. This section describes scenario settings that affect the generated code, as well as procedures for generating, compiling, and running generated code.



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the Java Code Generation video](#).

A complete list of all the videos demonstrating Stylus Studio's features is here: http://www.stylusstudio.com/xml_videos.html.

This section covers the following topics:

- “Scenario Settings” on page 402
- “Java Code Generation Settings” on page 404
- “How to Generate Java Code for XSLT” on page 404
- “Compiling Generated Code” on page 406
- “Deploying Generated Code” on page 408

Tip You can also generate Java code for XQuery. See “Generating Java Code for XQuery” on page 819.

Scenario Settings

Stylus Studio generates Java code based on the scenarios you have defined for an XSLT document. The following tables summarizes the scenario settings that have an effect on Java code generation.

Table 53. Scenario Settings that Affect Java Code Generation

<i>Tab</i>	<i>Comment</i>
General	The Java Code Generation wizard uses only the Source XML URL , and the Output URL field, if specified. All other properties on this page are ignored.
Parameter Values	Default parameter values and parameters in the form of XPath expressions are ignored.
Processor	You must use a Saxon processor. The Stylus Studio URI Resolver is also used to resolve non-standard URIs if you select the check box on the Processor page.
Profiling Options	Ignored.
Validation	Validation is optional. If you choose to validate XSLT output, the Java Code Generation wizard always uses the built-in Java processor, regardless of the validator you specify on the Validation tab. If you want to specify external schemas for validation purposes, click Use built-in validator . Note that the built-in Java processor is used for validation even in this case.
Post-process	Only post-processing using Apache FOP is specified in the generated code. Resulting PDF is written to the output URL.

Choosing Scenarios

You can generate Java code for one or more of the scenarios defined for a single XSLT document. By default, the Java Code Generation wizard selects only the current scenario for generation, as shown in [Figure 212](#).

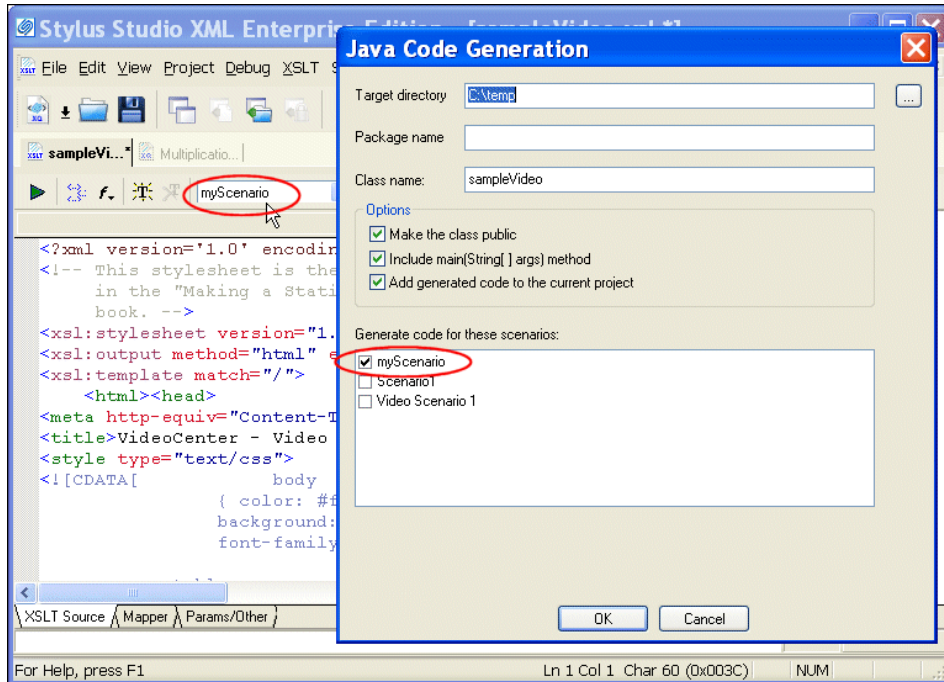


Figure 212. The Current Scenario Is Selected for Code Generation

The generated code includes a `setScenario` method for every scenario you select for code generation, as shown in the following example.

```
// Uncomment the setScenario call for the scenario you want the code to run.
// All other scenarios must be commented out.
app.setScenario_myScenario();
// app.setScenario_Scenario1();
// app.setScenario_Video_0032_Scenario_0032_1()
```

Only one `setScenario` method can be called at a time. Uncomment the `setScenario` method for the scenario you want the code to process, and make sure that all other `setScenario` methods are commented.

Java Code Generation Settings

When you generate Java code for an XSLT document, you need to specify

- The target directory in which you want the Java code created. `c:\temp\myJavaCode`, for example. If the directory you name does not exist, Stylus Studio creates it when you run the Java Code Generation wizard. The default is a `\sources` directory, which is created where you installed Stylus Studio when you generate the code, `c:\Program Files\Stylus Studio\sources`, for example.
- Optionally, a package name. If you specify a package name, this name is used for a subfolder created in the target directory you specify. If you specify `myPackage` as the package name, for example, the generated code is written to `c:\temp\myJavaCode\myPackage`. (Though optional, it is considered good practice to create a package name.)
- The class name. Stylus Studio also uses the class name for the `.java` file created by the Java Code Generation wizard. For example, if you provide the name `myClass`, Stylus Studio creates `c:\temp\myJavaCode\myPackage\myClass.java`.

In addition, you can specify whether or not you want to

- Create the class as a public class
- Include the `main(String[] args)` method
- Add the generated code to the current project

All of these options are selected by default.

Tip If you choose to add the generated code to the project, it creates a folder using the package name you specify and places the `.java` file in that folder. If you do not specify a package name, the `.java` file is added directly below the project root in the **Project** window.

How to Generate Java Code for XSLT

◆ **To generate Java code for XSLT:**

1. Define at least one scenario for the XSLT for which you want to generate Java code. The scenario must use the Saxon processor. See “[Scenario Settings](#)” on page 402 for more information.

2. Select **XSLT > Generate Java Code** from the Stylus Studio menu.
The **Generate Java Code** dialog box appears.

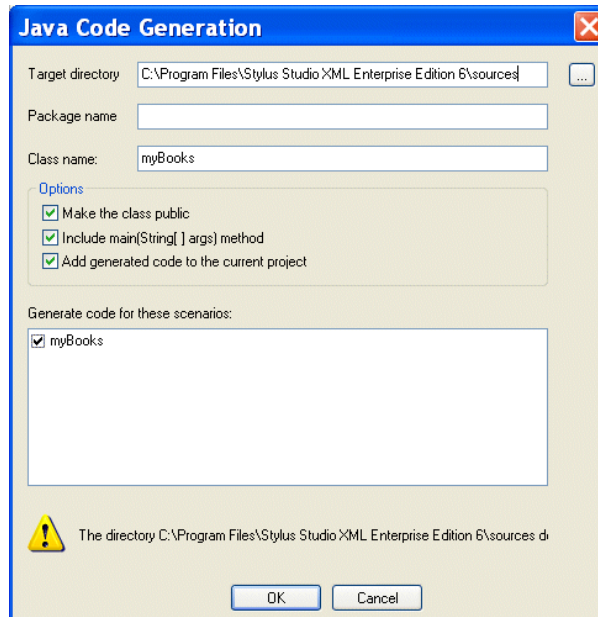


Figure 213. Java Code Generation Dialog Box

3. Specify the settings you want for the target directory, package and class names, and so on. See [“Java Code Generation Settings”](#) on page 404 if you need help with this step.
4. Select the scenarios for which you want to generate Java code.
5. Click **OK**.

Stylus Studio generates Java code for the XSLT. When the code generation is complete, the resulting file (*classname.java*) is opened in the Stylus Studio Java Editor.

Compiling Generated Code

In order to compile the Java code generated for XSLT, you need to make sure that the following JAR files are in the Stylus Studio classpath:

- CustomFileSystem.jar
- Saxon8.jar


These files are in in the \bin directory where you installed Stylus Studio.

How to Modify the Stylus Studio Classpath

◆ **To modify the Stylus Studio classpath:**

1. Select **Tool > Options** from the Stylus Studio menu.
2. Click **Java Virtual Machine** under **General**.
3. Click the browse button alongside the **Classpath** field.

The **Add Directory or JAR File to Classpath** dialog box appears.

4. Click the browse folders () button.

A new entry field appears in the **Locations** list box. Two buttons appear to the right of the entry field.

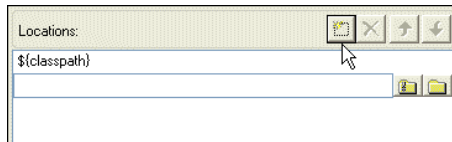



Figure 214. Entry Field for JAR File Classpath

5. Click the browse jar files button ().

Stylus Studio displays the **Browse for JAR Files** dialog box.

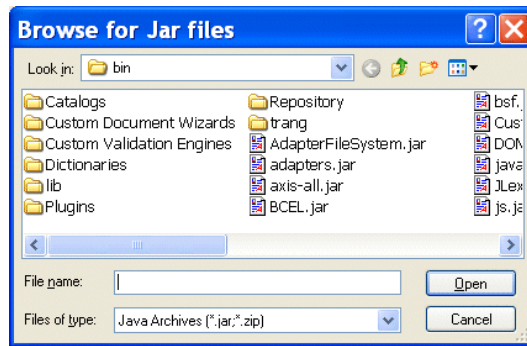



Figure 215. Browse for JAR Files Dialog Box


6. Navigate to the Stylus Studio \bin directory, select the CustomFileSystem.jar file, then click **Open**.
7. Repeat [Step 6](#), this time selecting the Saxon8.jar file.
8. Click **OK** to close the **Add Directory or JAR File to Classpath** dialog box.

How to Compile and Run Java Code in Stylus Studio

◆ To compile Java code in Stylus Studio:

1. Make sure the Java Editor is the active window.
2. Click the **Compile** button ().
Alternatives: Press Ctrl + F7, or select **Java > Compile** from the Stylus Studio menu. Stylus Studio compiles the Java code. Results are displayed in the **Output** window.

◆ To run Java code in Stylus Studio:

1. Make sure the Java Editor is the active window.
2. Click the **Run** button ().
Alternatives: Press Ctrl + F5, or select **Java > Run** from the Stylus Studio menu. If the code has not been compiled, Stylus Studio displays a prompt asking if you want to compile the code now. Otherwise, Stylus Studio runs the Java code. Results are displayed in the **Output** window.

Deploying Generated Code

If your XSLT uses built-in DataDirect XML Converters™ – to convert CSV or EDI to XML, for example – you need to purchase licenses for the DataDirect XML Converters you wish to use if you wish to deploy your code in any environment on a machine (such as a test or application server) that does not have a license for the DataDirect XML Converters. Licenses for DataDirect XML Converters are purchased separately from Stylus Studio 2007 XML Enterprise Suite.

Write Stylus Studio at stylusstudio@stylusstudio.com, or call 781.280.4488 for more information.

XSLT Instructions Quick Reference

This section provides a quick reference for the XSLT instructions supported by the Stylus Studio XSLT processor.

For more information on

- XSLT 1.0, go to <http://www.w3.org/TR/xslt>
- XSLT 2.0, go to <http://www.w3.org/TR/xslt/20>

This section covers the following instructions:

- [xsl:apply-imports](#) on page 410
- [xsl:apply-templates](#) on page 410
- [xsl:attribute](#) on page 411
- [xsl:attribute-set](#) on page 412
- [xsl:call-template](#) on page 414
- [xsl:character-map](#) on page 414
- [xsl:choose](#) on page 417
- [xsl:comment](#) on page 418
- [xsl:copy](#) on page 418
- [xsl:copy-of](#) on page 419
- [xsl:decimal-format](#) on page 420
- [xsl:element](#) on page 421
- [xsl:fallback](#) on page 422
- [xsl:for-each](#) on page 422
- [xsl:for-each-group](#) on page 424

- [xsl:function](#) on page 425
- [xsl:if](#) on page 426
- [xsl:import](#) on page 427
- [xsl:import-schema](#) on page 427
- [xsl:include](#) on page 429
- [xsl:key](#) on page 430
- [xsl:message](#) on page 431
- [xsl:namespace-alias](#) on page 432
- [xsl:number](#) on page 432
- [xsl:otherwise](#) on page 433
- [xsl:output](#) on page 433
- [xsl:output-character](#) on page 436
- [xsl:param](#) on page 436
- [xsl:preserve-space](#) on page 438
- [xsl:processing-instruction](#) on page 438
- [xsl:sequence](#) on page 439
- [xsl:sort](#) on page 439
- [xsl:strip-space](#) on page 441
- [xsl:stylesheet](#) on page 442
- [xsl:template](#) on page 442
- [xsl:text](#) on page 444
- [xsl:transform](#) on page 445
- [xsl:value-of](#) on page 445
- [xsl:variable](#) on page 446
- [xsl:when](#) on page 447
- [xsl:with-param](#) on page 447

xsl:apply-imports

Invokes overridden template rules.

Stylus Studio does not support the `xsl:apply-imports` instruction.

xsl:apply-templates

Selects source nodes for processing.

Format

```
<xsl:apply-templates [select="pattern"] [mode="qname"]>
  [<xsl:sort/>]
  [<xsl:with-param/>]
</xsl:apply-templates>
```

Description

If you specify the `select` attribute, specify a pattern that resolves to a set of source nodes. For each source node in this set, the XSLT processor searches for a template that matches the node. When it finds a matching template, it instantiates it and uses the node as the context node. For example:

```
<xsl:apply-templates select="/bookstore/book">
```

When the XSLT processor executes this instruction, it constructs a list of all nodes that match the pattern in the `select` attribute. For each node in the list, the XSLT processor searches for the template whose `match` pattern best matches that node.

If you do not specify the `select` attribute, the XSLT processor uses the default pattern, `"node()"`, which selects all child nodes of the current node.

If you specify the `mode` attribute, the selected nodes are matched only by templates with a matching `mode` attribute. The value of `mode` must be a qualified name or an asterisk (*). If you specify an asterisk, it means continue the current mode, if any, of the current template.

If you do not specify a `mode` attribute, the selected nodes are matched only by templates that do not specify a `mode` attribute.

By default, the new list of source nodes is processed in document order. However, you can use the `xsl:sort` instruction to specify that the selected nodes are to be processed in a different order. See [“xsl:sort”](#) on page 439.

Tip You can create an `xsl:apply-templates` element automatically using the XSLT mapper.

Example

In the previous example, the XSLT processor searches for a template that matches `/bookstore/book`. The following template is a match:

```
<xsl:template match="book">
  <tr>
    <td><xsl:value-of select="title"/></td>
    <td><xsl:value-of select="author"/></td>
    <td><xsl:value-of select="price"/></td>
  </tr>
</xsl:template>
```

The XSLT processor instantiates this template for each book element.

xsl:attribute

Creates an attribute.

Format

```
<xsl:attribute name="qualified_name">
  attribute_value
</xsl:attribute>
```

Description

You can specify the `xsl:attribute` instruction in the

- Contents of a stylesheet element that creates a result element
- Contents of an `xsl:attribute-set` instantiation

In a stylesheet element that creates a result element, the `xsl:attribute` instruction causes an attribute to be added to the created result element.

The prefix part of the name attribute value becomes the prefix for the attribute you are creating. The local part of the name attribute value becomes the local name of the attribute you are creating.

The XSLT processor interprets the name attribute as an attribute value template. The string that results from instantiating the attribute value template must be a qualified name. If it is not, the XSLT processor reports an error.

The result of instantiating the content of the `xsl:attribute` instruction is used as the value of the created attribute. It is an error if instantiating this content generates anything other than characters.

If you add an attribute to an element and that element already has an attribute with the same expanded name, the attribute you are creating replaces the existing attribute.

Example

```
<xsl:attribute name="library:ISBN"
  namespace="http://www.library.org/namespaces/library">
  1-2222-333-4
</xsl:attribute>
```

If this instruction is inside a book element, the resulting book element would include the following attribute:

```
library:ISBN="1-2222-333-4"
```

The XSLT processor reports an error if you try to do any of the following:

- Add an attribute to a node that is not an element.
- Add an attribute to an element that already has child nodes.
- Create anything other than characters during instantiation of the contents of the `xsl:attribute` element.

xsl:attribute-set

Defines a named set of attributes.

Format

```
<xsl:attribute-set name="set_name">
  <xsl:attribute name="attr_name">attr_value</xsl:attribute>
  <xsl:attribute name="attr_name">attr_value</xsl:attribute>
  ...
</xsl:attribute-set>
```

Description

The name attribute specifies the name of the attribute set. This must be a qualified name. The contents of the `xsl:attribute-set` element consists of zero or more `xsl:attribute` elements. Each `xsl:attribute` element specifies an attribute in the set.

To use an attribute set, specify the `use-attribute-sets` attribute in one of the following elements:

- `xsl:element`
- `xsl:copy`

- `xsl:attribute-set`

The value of the `use-attribute-sets` attribute is a white-space-separated list of names of attribute sets. When you specify the use of an attribute set, it is equivalent to adding an `xsl:attribute` element for each attribute in each named attribute set to the beginning of the contents of the element in which you specify the `use-attribute-sets` attribute.

An attribute set cannot include itself. In other words, if attribute set A specifies the `use-attribute-sets` attribute, the list of attribute sets to use cannot include attribute set A.

You can also specify an attribute set in an `xsl:use-attribute-sets` attribute on a literal result element. The value of the `xsl:use-attribute-sets` attribute is a white-space-separated list of names of attribute sets. The `xsl:use-attribute-sets` attribute has the same effect as the `use-attribute-sets` attribute on `xsl:element` with one additional rule. The additional rule is that attributes specified on the literal result element itself are treated as if they were specified by `xsl:attribute` elements before any actual `xsl:attribute` elements but after any `xsl:attribute` elements implied by the `xsl:use-attribute-sets` attribute.

Thus, for a literal result element, attributes from attribute sets named in an `xsl:use-attribute-sets` attribute are added first, in the order listed in the attribute. Next, attributes specified on the literal result element are added. Finally, any attributes specified by `xsl:attribute` elements are added. Since adding an attribute to an element replaces any existing attribute of that element with the same name, this means that attributes specified in attribute sets can be overridden by attributes specified on the literal result element itself.

The template within each `xsl:attribute` element in an `xsl:attribute-set` element is instantiated each time the attribute set is used. It is instantiated using the same current node and current node list as is used for instantiating the element bearing the `use-attribute-sets` or `xsl:use-attribute-sets` attribute. However, it is the position in the stylesheet of the `xsl:attribute` element rather than of the element bearing the `use-attribute-sets` or `xsl:use-attribute-sets` attribute that determines which variable bindings are visible. Consequently, only variables and parameters declared by top-level `xsl:variable` and `xsl:param` elements are visible.

The XSLT processor merges multiple definitions of an attribute set with the same expanded name. If there are two attribute sets with the same expanded name that both contain the same attribute, the XSLT processor chooses the attribute definition that was specified last in the stylesheet.

Example

The following example creates a named attribute set, `title-style`, and uses it in a template rule:

```
<xsl:template match="chapter/heading">
  <fo:block quadding="start" xsl:use-attribute-sets="title-style">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:attribute-set name="title-style">
  <xsl:attribute name="font-size">12pt</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
</xsl:attribute-set>
```

xsl:call-template

Instantiates a named template.

Format

```
<xsl:call-template name="template_name">
  [<xsl:with-param/>]
</xsl:apply-templates>
```

Description

The `name` attribute is required and the value must be a qualified name. It specifies the name of the template you want to instantiate. The template you want to instantiate must specify the `name` attribute with a value identical to `template_name`.

Unlike the `xsl:apply-templates` instruction, the `xsl:call-template` instruction does not change the current node.

Tip You can create an `xsl:call-template` element automatically using the XSLT mapper.

xsl:character-map

Declares a character map defined by a unique name. A stylesheet cannot contain two or more character maps with the same name. Character maps are supported by XSLT 2.0 only.

Tip Character maps are an alternative to defining character entities using a DTD, which was required by XSLT 1.0.

Format

```
<xsl:character-map
  name = QName
  [use-character-maps = Qnames]>
  [<xsl:output-character> ...]
</xsl:character-map>
```

Description

A *character map* allows a specific character appearing in a text or attribute node in the final result tree to be substituted by a specified string of characters during serialization. The character map that is supplied as a parameter to the serializer is determined from the `xsl:character-map` elements referenced from the `xsl:output` declaration for the selected output definition.

Character/string mappings can be defined in the body of the `xsl:character-map` element using one or more `xsl:output-character` elements, or they can be defined in an external character map referenced using the optional `use-character-maps` attribute. If the character map references multiple external character maps, separate each character map's *Qname* with a space.

An output definition, after recursive expansion of character maps referenced via its `use-character-maps` attribute, may contain several mappings for the same character. In this situation, the last character mapping takes precedence.

If a character is mapped, it is not subjected to XML or HTML escaping.

Example

This example shows a composite character map – one constructed using both internally defined character mappings (`xsl:output-character`), and references to externally defined character maps (the `use-character-maps` attribute):

```
<xsl:output name="htmlDoc" use-character-maps="htmlDoc" />
<xsl:character-map name="htmlDoc"
  use-character-maps="html-chars doc-entities windows-format" />
<xsl:character-map name="html-chars"
  use-character-maps="latin1 ..." />
<xsl:character-map name="latin1">
  <xsl:output-character character=" " string="&nbsp;" />
  <xsl:output-character character="&" string="&iexcl;" />
  ...
</xsl:character-map>
<xsl:character-map name="doc-entities">
  <xsl:output-character character="&" string="&t-and-c;" />
  <xsl:output-character character="&" string="&chap1;" />
  <xsl:output-character character="&" string="&chap2;" />
  ...
</xsl:character-map>
<xsl:character-map name="windows-format">
  <!-- newlines as CRLF -->
  <xsl:output-character character="
" string="
" />

  <!-- tabs as three spaces -->
  <xsl:output-character character="	" string="   " />

  <!-- images for special characters -->
  <xsl:output-character character=""
    string="&lt;img src='special1.gif' /&gt;" />
  <xsl:output-character character=""
    string="&lt;img src='special2.gif' /&gt;" />
  ...
</xsl:character-map>
```

xsl:choose

Selects one template to instantiate from a group of templates.

Format

```
<xsl:choose>
  <xsl:when test="expression1">
    template_body
  </xsl:when>
  [<xsl:when test="expression2">
    template_body
  </xsl:when>] ...
  [<xsl:otherwise>
    template_body
  </xsl:otherwise>]
</xsl:choose>
```

Description

An `xsl:choose` element contains one or more `xsl:when` elements followed by zero or one `xsl:otherwise` element. Each `xsl:when` element contains a required `test` attribute, whose value is an expression. Each `xsl:when` and `xsl:otherwise` element contains a template.

When the XSLT processor processes an `xsl:choose` element, it starts by evaluating the expression in the first `xsl:when` element. The XSLT processor converts the result to a Boolean value. If the result is `true`, the XSLT processor instantiates the template contained by that `xsl:when` element. If the result is `false`, the XSLT processor evaluates the expression in the next `xsl:when` element.

The XSLT processor instantiates the template of only the first `xsl:when` element whose test expression evaluates to `true`. If no expressions evaluate to `true` and there is an `xsl:otherwise` element, the XSLT processor instantiates the template in the `xsl:otherwise` element.

If no expressions in `xsl:when` elements are `true` and there is no `xsl:otherwise` element, the `xsl:choose` element has no effect.

Tip You can create an `xsl:choose` element automatically using the XSLT mapper.

xsl:comment

Adds a comment node to the result tree.

Format

```
<xsl:comment>  
  comment_text  
</xsl:comment>
```

Description

The XSLT processor instantiates the contents of the instruction to generate the text of the new comment.

The XSLT processor reports an error if instantiating the contents of the `xsl:comment` instruction creates anything other than characters, or if the resulting string contains the substring "--" or ends with "-".

Example

The following instruction creates a comment in the result document:

```
<xsl:comment>Unique Irish band</xsl:comment>
```

The comment is

```
<!--Unique Irish band-->
```

xsl:copy

Adds a copy of the current node to the result tree.

Format

```
<xsl:copy>copy_contents</xsl:copy>
```

Description

The copy includes the current node's namespace information but does not include the current node's attributes or children. The contents of the `xsl:copy` element is a template for the attributes and children of the node being created. If the current node cannot have

attributes or children (that is, if it is an attribute, text, comment, or processing instruction node), the content of the instruction is ignored.

If the current node is the root node, the XSLT processor does not create a root node. Instead, it uses *copy_contents* as a template.

Example

Following is an example from the W3C XSLT Recommendation. It generates a copy of the source document.

```
<xsl:template match="@* | node() ">
  <xsl:copy>
    <xsl:apply-templates select="@* | node() " />
  </xsl:copy>
</xsl:template>
```

xsl:copy-of

Inserts the value of an expression into the result tree, without first converting it to a string.

Format

```
<xsl:copy-of select = "expression" />
```

Description

The required `select` attribute contains an expression. When the result of evaluating the expression is a result tree fragment, the XSLT processor copies the complete fragment into the result tree. When the result is a node set, the XSLT processor copies all nodes in the set, together with their contents, in document order into the result tree. When the result is of any other type, the XSLT processor converts the result to a string and then inserts the string into the result tree in the same way that `xsl:value-of` does.

Tip You can also use `xsl:sequence` to add atomic values to a sequence. See [xsl:sequence](#) on page 439 for more information.

xsl:decimal-format

Declares a decimal format.

Format

```
<xsl:decimal-format
  name = qname
  decimal-separator = char
  grouping-separator = char
  infinity = string
  minus-sign = char
  NaN = string
  percent = char
  per-mille = char
  zero-digit = char
  digit = char
  pattern-separator = char />
```

Description

The `xsl:decimal-format` instruction declares a decimal format, which controls the interpretation of a format pattern that is used by the `format-number()` function.

If there is a `name` attribute, the element declares a named decimal format. Otherwise, it declares the default decimal format. The value of the `name` attribute is a qualified name.

The other attributes on `xsl:decimal-format` correspond to the methods on the JDK `DecimalFormatSymbols` class. For each `get/set` method pair, there is an attribute defined for the `xsl:decimal-format` instruction.

The following attributes control the interpretation of characters in the format pattern and specify characters that can appear in the result of formatting the number:

- `decimal-separator` specifies the character used for the decimal sign; the default value is the dot character (`.`).
- `grouping-separator` specifies the character used as a grouping (for example, thousands) separator; the default value is the comma character (`,`).
- `percent` specifies the character used as a percent sign; the default value is the percent character (`%`).
- `per-mille` specifies the character used as a per mille sign; the default value is the Unicode per mille character (`#x2030`).
- `zero-digit` specifies the character used as the digit zero; the default value is the digit zero (`0`).

The following attributes control the interpretation of characters in the format pattern:

- `digit` specifies the character used for a digit in the format pattern; the default value is the number sign character (#).
- `pattern-separator` specifies the character used to separate positive and negative subpatterns in a pattern; the default value is the semicolon character (;).

The following attributes specify characters or strings that can appear in the result of formatting the number:

- `infinity` specifies the string used to represent infinity; the default value is the string "Infinity".
- `minus-sign` specifies the character used as the default minus sign; the default value is the hyphen (minus) character (-, #x2D).
- `NaN` specifies the string used to represent the NaN value; the default value is the string "NaN".

xsl:element

Adds an element to the result tree.

Format

```
<xsl:element name="qualified_name">  
  element_contents  
</xsl:element>
```

Description

The XSLT processor uses the contents of the `xsl:element` instruction as a template for the attributes and contents of the new element.

The prefix part of the `name` attribute becomes the prefix for the element you are creating. The local part of the `name` attribute becomes the local name of the element you are creating.

The XSLT processor interprets the `name` attribute as an attribute value template. The string that results from instantiating the attribute value template must be a qualified name. If it is not, the XSLT processor reports an error.

Example

```
<xsl:element name="audio:CD">  
  <xsl:element name="audio:title">Celtic Airs</xsl:element>  
  <xsl:element name="audio:artist">Chieftains</xsl:element>  
</xsl:element>
```

The result of this instruction looks like the following:

```
<audio:CD>  
  <audio:title>Celtic Airs</audio:title>  
  <audio:artist>Chieftains</audio:artist>  
</audio:CD>
```

xsl:fallback

Normally, instantiating an `xsl:fallback` element does nothing. However, when an XSLT processor performs fallback for an instruction element, if the instruction element has one or more `xsl:fallback` children, then the content of each of the `xsl:fallback` children must be instantiated in sequence; otherwise, an error is signaled. The content of an `xsl:fallback` element is a template.

xsl:for-each

Selects a set of nodes in the source document and instantiates the contained template once for each node in the set.

Format

```
<xsl:for-each select="pattern">  
  [<xsl:sort[select="expression"] [optional_attribute]/>]  
  template_body  
</xsl:for-each>
```

Description

The `select` attribute is required and the pattern must evaluate to a node set. The XSLT processor instantiates the embedded template with the selected node as the current node and with a list of all selected nodes as the current node list.

By default, the new list of source nodes is processed in document order. However, you can use the `xsl:sort` instruction to specify that the selected nodes are to be processed in a different order. See “[xsl:sort](#)” on page 439.

The `xsl:for-each` instruction is useful when the result document has a regular, known structure. When you know that you want to instantiate the same template for each node in the current node list, the `xsl:for-each` instruction eliminates the need to find a template that matches each node.

Tip You can create an `xsl:for-each` element automatically using the XSLT mapper.

Example

For example, suppose your source document includes the following XML:

```
<books>
  <author>
    <name>Sara Peretsky</name>
    <booktitle>Bitter Medicine</booktitle>
    <booktitle>Killing Orders</booktitle>
  </author>
  <author>
    <name>Dick Francis</name>
    <booktitle>Reflex</booktitle>
    <booktitle>Proof</booktitle>
    <booktitle>Nerve</booktitle>
  </author>
</books>
```

The following stylesheet creates an HTML document that contains a list of authors. Each author is followed by the titles of the books the author wrote. It does not matter how many authors there are nor how many titles are associated with each author. The stylesheet uses the `xsl:for-each` instruction to process each author and to process each title associated with each author.

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match = "/">
  <html>
    <head><title>Authors and Their Books</title></head>
    <body>
      <xsl:for-each select = "books/author">
        <p>
          <xsl:value-of select = "name"/>
          <br>
          <xsl:for-each select = "booktitle">
            <xsl:value-of select = "."/>
          <br>
        </xsl:for-each>
        </p>
      </xsl:for-each>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

The result document looks like this:

```
<html>
<head>
<title>Authors and Their Books</title>
</head>
<body>
<p>
Sara Peretsky<br>
Bitter Medicine<br>
Killing Orders<br>
</p>
<p>
Dick Francis<br>
Reflex<br>
Proof<br>
Nerve<br>
</p>
</body>
</html>
```

xsl:for-each-group

Allocates the items in an input sequence into groups of items (that is, it establishes a collection of sequences) based either on common values of a grouping key, or on a pattern that the initial or final node in a group must match.

Format

```
<xsl:for-each-group
  select = expression
  [group-by = expression]
  [group-adjacent = expression]
  [group-starting-with = pattern]
  [group-ending-with = pattern]
  [collation = { uri }]>
  <!-- Content: (xsl:sort*, sequence-constructor) -->
</xsl:for-each-group>
```

Description

The `xsl:for-each-group` element is an instruction that can be used anywhere within a sequence constructor. The sequence of items to be grouped is referred to as a *population*. A group cannot be empty. If the population is zero (that is, empty), the number of groups is zero. How items are assigned to groups is determined by the `group-by`, `group-adjacent`, `group-starting-with`, and `group-ending-with` attributes.

xsl:function

Allows the creation of user-defined stylesheet function that can be called from any XPath expression within the stylesheet in which the function is defined. This instruction is supported in XSLT 2.0 only.

Format

```
<xsl:function name="Qname" as="sequence type" [override="yes" | "no"]>  
  function_body  
</xsl:function>
```

Description

The value of the `name` attribute, *Qname*, is a qualified name and takes the form *prefix:name*. The prefix is required in order to avoid possible conflicts with any functions in the default function namespace. The prefix cannot refer to a reserved namespace.

The *function_body* contains zero or more “[xsl:param](#)” on page 436 elements that specify the formal arguments of the function. These `xsl:param` elements are followed by a sequence constructor that defines the value to be returned by the function. The `xsl:param` elements within an `xsl:function` element must be empty; they cannot have a `select` attribute because they must be specified.

An `xsl:function` declaration can only appear as a top-level element in a stylesheet.

Example

Here is an example from the W3C XSLT Working Draft of a simple function that reverses the order of the words in a sentence.

```
<xsl:transform
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:str="http://example.com/namespace"
  version="2.0"
  exclude-result-prefixes="str">

  <xsl:function name="str:reverse" as="xs:string">
    <xsl:param name="sentence" as="xs:string"/>
    <xsl:sequence
      select="if (contains($sentence, ' '))
              then concat(str:reverse(substring-after($sentence, ' ')),
                          ' ',
                          substring-before($sentence, ' '))
              else $sentence"/>
  </xsl:function>

  <xsl:template match="/">
    <output>
      <xsl:value-of select="str:reverse('DOG BITES MAN')"/>
    </output>
  </xsl:template>
</xsl:transform>
```

xsl:if

Conditionally instantiates the contained template body.

Format

```
<xsl:if test = "expression">
  template_body
</xsl:if>
```

Description

The XSLT processor evaluates the expression and converts the result to a Boolean value. If the result is `true`, the XSLT processor instantiates *template_body*. If the result is `false`, the `xsl:if` element has no effect.

Example

This following example formats a group of names as a comma-separated list:

```
<xsl:template match="namelist/name">
  <xsl:value-of select="." />
  <xsl:if test="not(position()=last())">, </xsl:if>
</xsl:template>
```

If you want the XSLT processor to choose which template to instantiate from several possibilities, specify the `xsl:choose` instruction. See [“xsl:choose”](#) on page 417.

xsl:import

Imports a stylesheet into the stylesheet containing this instruction.

Format

```
<xsl:import href="stylesheet_path">
```

stylesheet_path specifies the stylesheet you want to import. Specify a URL, a relative path, or a DOS-style path.

Description

An XSLT stylesheet can import another XSLT stylesheet by using an `xsl:import` instruction. Importing a stylesheet is the same as including it, except that definitions and template rules in the importing stylesheet take precedence over template rules and definitions in the imported stylesheet.

The `xsl:import` element is only allowed as a top-level element. The `xsl:import` element children must precede all other element children of an `xsl:stylesheet` element, including any `xsl:include` element children. When `xsl:include` is used to include a stylesheet, any `xsl:import` elements in the included document are moved up in the including document to after any existing `xsl:import` elements in the including document.

When you use the `xsl:import` instruction, templates have an `importance` property.

xsl:import-schema

Identifies schema components (top-level type definitions and top-level element and attribute declarations) that need to be available statically, that is, before any source

document is available. Allows you to extend XSLT built-in types with the types defined in the imported XML Schema.

Format

```
<xsl:import-schema
  namespace = uri-reference
  schema-location = uri-reference>
<!-- Content: xs:schema -->
</xsl:import-schema>
```

Description

The `xsl:import-schema` declaration identifies a namespace containing the names of the components to be imported (or indicates that components whose names are in no namespace are to be imported). The effect is that the names of top-level element and attribute declarations and type definitions from this namespace (or non-namespace) become available for use within XPath expressions in the stylesheet, and within other stylesheet constructs such as the type and as attributes of various XSLT elements.

The same schema components are available in all stylesheet modules; importing components in one stylesheet module makes them available throughout the stylesheet.

The `namespace` and `schema-location` elements are optional. The `namespace` attribute indicates that a schema for the given namespace is required by the stylesheet. This information may be enough on its own to enable an implementation to locate the required schema components. The `namespace` attribute may be omitted to indicate that a schema for names in no namespace is being imported. The zero-length string is not a valid namespace URI, and is therefore not a valid value for the `namespace` attribute.

The `schema-location` attribute is a URI Reference that describes where a schema document or other resource containing the required definitions may be found. It is likely that a schema-aware XSLT processor will be able to process a schema document found at this location.

The use of a namespace in an `xsl:import-schema` declaration does not by itself associate any namespace prefix with the namespace. If names from the namespace are used within the stylesheet module then a namespace declaration must be included in the stylesheet module, in the usual way.

You can also define an inline schema document using the `xs:schema` element as a child of `xsl:import-schema`. An inline schema document has the same status as an external schema document, in the sense that it acts as a hint for a source of schema components in the

relevant namespace. To ensure that the inline schema document is always used, it is advisable to use a target namespace that is unique to this schema document.

Example

The following example shows an inline schema document defined using the `xs:schema` subelement. This schema declares a simple type `local:yes-no`, which the stylesheet then uses in the declaration of a variable. The example assumes the namespace declaration `xmlns:local="http://localhost/ns/yes-no"`.

```
<xsl:import-schema>
  <xs:schema targetNamespace="http://localhost/ns/yes-no">
    <xs:simpleType name="local:yes-no">
      <xs:restriction base="xs:string">
        <xs:enumeration value="yes"/>
        <xs:enumeration value="no"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:schema>
</xs:import-schema>

<xsl:variable name="condition" select="'yes'" as="local:yes-no"/>
```

xsl:include

Specifies an XSLT stylesheet that is included in and combined with the stylesheet that specifies `xsl:include`.

Format

```
<xsl:include href="stylesheet_path">
```

stylesheet_path specifies the stylesheet you want to import. Specify a URL, a relative path, or a DOS-style path.

Description

The `xsl:include` instruction must be a child of an `xsl:stylesheet` element. The XSLT processor effectively replaces the `xsl:include` instruction with the children of the root `xsl:stylesheet` element of the included stylesheet. If the root element of the included stylesheet is a literal result element, the XSLT processor effectively replaces the `xsl:include` instruction with the following new element whose only child is that literal result element:

```
<xsl:template match="/">
```

A stylesheet cannot include itself directly or indirectly.

xsl:key

Declares a key for a document.

Format

```
<xsl:key name="qname"  
  match = "pattern"  
  use = "use" />
```

Description

Keys provide a way to work with documents that contain an implicit cross-reference structure. A stylesheet declares a key for a document with the `xsl:key` instruction.

The `xsl:key` instruction must be a top-level element. It has no contents, but it specifies three attributes.

Replace *qname* with the name of the key. You must specify a qualified name.

Replace *pattern* with a pattern that identifies one or more nodes that have this key. In other words, the nodes in the document that match the pattern are included in the key. The default is `node()`.

Replace *use* with an expression that you want to use for the key values. The XSLT processor evaluates the expression once for each node in the set identified by *pattern*.

Each key name represents a separate, independent set of identifiers. Each node included in a key is associated with a set of string key values. These values result from evaluating the *use* expression with that node as the current node.

A document can contain multiple keys with the same node and the same key name, but with different key values. A document can contain multiple keys with the same key name and value, but with different nodes. In other words:

- A node can be included in more than one key.
- For a given key, a key value can be associated with more than one node.
- The same key value can be associated with different nodes in different keys.

The value of a key can be an arbitrary string. It need not be a name.

Use the XSLT `key()` function to retrieve the list of nodes included in a given key that have given key values. See [“Finding an Element with a Particular Key”](#) on page 703.

You cannot specify multiple declarations for the same key in a stylesheet. Stylus Studio expects to remove this restriction in a future release.

xsl:message

Sends a message in a way that is dependent on the XSLT processor.

Format

```
<xsl:message terminate="yes" | "no">  
  <!-- Content: template -->  
</xsl:message>
```

Description

The content of the `xsl:message` instruction is a template. If the value of the `terminate` attribute is `yes`, the XSLT processor instantiates the template to create text. The processor aborts stylesheet processing and sends the text as part of the error message that indicates that stylesheet processing has terminated.

The default value of the `terminate` attribute is `no`. If you specify `terminate="no"` or if you do not specify the `terminate` attribute, the XSLT processor displays the message in the Stylus Studio **Output Window** but does not terminate the process.

xsl:namespace-alias

Causes the namespace URI to be changed in the output.

Format

```
<xsl:namespace-alias
  stylesheet-prefix = prefix | "#default"
  result-prefix = prefix | "#default" />
```

Description

Declares that one namespace URI is an alias for another namespace URI. When a literal namespace URI has been declared to be an alias for another namespace URI, then the namespace URI in the result tree is the namespace URI that the literal namespace URI is an alias for, instead of the literal namespace URI itself.

xsl:number

Inserts a formatted number into the result tree.

Format

```
<xsl:number
  [level = "single" | "multiple" | "any"]
  [count = pattern]
  [from = pattern]
  [value = number-expression]
  [format = {string}]
  [lang = {nmtoken}]
  [letter-value = {"alphabetic" | "traditional"}]
  [grouping-separator = {char}]
  [grouping-size = {number}] />
```

Description

You can use the `value` attribute to specify an expression for the number to be inserted. The XSLT processor evaluates the expression. The resulting object is converted to a number as if by a call to the `number()` function. The processor rounds the number to an integer and then uses the specified attributes to convert it to a string. The value of each attribute is interpreted as an attribute value template. After conversion, the resulting string is inserted in the result tree.

The following attributes control how the current node is to be numbered:

- The `level` attribute specifies what levels of the source tree should be considered. The default is `single`.
- The `count` attribute is a pattern that specifies what nodes should be counted at those levels.
- The `from` attribute is a pattern that specifies where counting starts.
- The `value` attribute can specify an expression that represents the number you want to insert. If no `value` attribute is specified, the XSLT processor inserts a number based on the position of the current node in the source tree.
- The `format` attribute specifies the format for each number in the list. The default is `1`.
- The `lang` attribute specifies which language's alphabet is to be used.
- The `letter-value` attribute distinguishes between the numbering sequences that use letters.
- The `grouping-separator` attribute specifies the separator used as a grouping (for example, thousands) separator in decimal numbering sequences.
- The `grouping-size` attribute specifies the size of the grouping. Normally, this is `3`.

Example

The following example numbers a sorted list:

```
<xsl:template match="items">
  <xsl:for-each select="item">
    <xsl:sort select="."/>
    <p>
      <xsl:number value="position()" format="1. "/>
      <xsl:value-of select="."/>
    </p>
  </xsl:for-each>
</xsl:template>
```

xsl:otherwise

See “[xsl:choose](#)” on page 417.

xsl:output

Specifies the output for the result tree.

Format

```
<xsl:output attribute_list />
```

Description

The `xs1:output` instruction specifies how you want the result tree to be output. However, if you use the XSLT processor to format the result as a string, or to generate DOM nodes, the `xs1:output` instruction has no effect.

If you specify the `xs1:output` instruction, the XSLT processor outputs the result tree according to your specification. If you specify it, the `xs1:output` instruction must be a top-level element.

The attribute list can include the `method` attribute. The `method` attribute identifies the overall method you want the XSLT processor to use to output the result tree. The value must be `xml`, `html`, or `text`.

- `xml` formats the result tree as XML.
- `html` formats the result tree as HTML. The stylesheet applies special formatting rules for empty tags, binary attributes, and character escaping, among other things. The values of the attributes named `href` and `src` are URL encoded.
- `text` concatenates the text nodes in the result tree. The concatenated string does not include any tags.

Note that the XSLT processor formats the results of applying the stylesheet. If your stylesheet generates XML or HTML that does not follow all syntax rules, the XSLT processor does not do anything to fix this. For example, if a stylesheet generates multiple root elements, the XSLT processor neither fixes this nor generates an error. You receive a string, and it is only upon examination or use of the string that you would learn that it is not well-formed XML.

If you do not specify an `xs1:output` instruction that includes the `method` attribute, the XSLT processor chooses a default as follows:

- `html` is the default output method if the name of the first element child of the root node is `html`.
- `text` is the default output method if the root node has no element child nodes.
- `xml` is the default output method in all other cases.

The other attributes that you can specify in `attribute_list` provide parameters for the output method. You can specify the following attributes:

- `doctype-public` specifies the public identifier to be used in the document type declaration.
- `doctype-system` specifies the system identifier to be used in the document type declaration.

- `encoding` specifies the preferred character encoding that the XSLT processor should use to encode sequences of characters as sequences of bytes.
- `indent` specifies whether the XSLT processor can add additional white space when outputting the result tree. The value must be `yes` or `no`.
- `media-type` specifies the media type (MIME content type) of the data that results from outputting the result tree. Do not explicitly specify the `charset` parameter. Instead, when the top-level media type is `text`, add a `charset` parameter according to the character encoding actually used by the output method.
- `omit-xml-declaration` specifies whether the XSLT processor should omit or output an XML declaration. The value must be `yes` or `no`. If you do not specify this attribute, whether or not the output contains an XML declaration depends on the output method.
 - If the output method is `html`, the XSLT processor does not insert an XML declaration.
 - If the output method is `xml`, the XSLT processor inserts an XML declaration.The XSLT processor ignores this attribute when the output method is `text`.
- `standalone` specifies whether the XSLT processor should output a stand-alone document declaration. The value must be `yes` or `no`.
- `use-character-map` specifies the name, if any, of the character map you want to use for the output. A character map substitutes characters based on character/string mappings declared in the `xsl:character-map` element.

A stylesheet can include multiple `xsl:output` elements. The XSLT processor effectively merges multiple `xsl:output` elements into one `xsl:output` element. If there are multiple values for the same attribute, the XSLT processor uses the last specified value.

In this release, the XSLT processor ignores the following attributes:

- `cdata-section-elements` specifies a list of the names of elements whose text node children should be output using CDATA sections.
- `version` specifies the version of the output method.

xsl:output-character

Declares character/string mappings used by the [xsl:character-map](#) declaration. `xsl:output-character` is supported in XSLT 2.0 only.

Format

```
<xsl:output-character
  character = char
  string = string />
```

Description

The character map that is passed as a parameter to the serializer contains a mapping for the character specified in the `character` attribute to the string specified in the `string` attribute.

Example

See [xsl:character-map](#) on page 414.

xsl:param

Declares a parameter for a stylesheet or template, and specifies a default value for the parameter.

Format

```
<xsl:param name="parameter_name"
  [select = "expression1"]
  [expr = "expression2"]>
  [template_body]
</xsl:param>
```

Description

The `xsl:param` instruction declares a parameter and specifies its default value. Another value can be passed to this parameter when the template or stylesheet that contains this `xsl:param` instruction is invoked.

The `xsl:param` element must be a child of either an `xsl:stylesheet` or `xsl:template` element.

The `name` attribute is required, and it must be a string. The value of the `name` attribute is a qualified name.

The value that you bind to a parameter can be an object of any of the types that are returned by expressions. You can specify the value of the parameter in several ways:

- Specify the `select` attribute. The value of the `select` attribute must be an expression. The XSLT processor evaluates the expression, and the result is the default value of the parameter. If you specify the `select` attribute, the XSLT processor ignores any value you might specify for the `expr` attribute, and also ignores any contents of `xsl:param`.
- Specify the `expr` attribute. The `expr` attribute allows computation of an expression. For example:

```
<xsl:param name="query" expr="//VEHICLE[MAKE='{ $make } ' ]"/>
```

The XSLT processor interprets the value of the `expr` attribute as an attribute value template and uses the resulting string as if it were the value of the `select` attribute. If you specify the `expr` attribute, the XSLT processor ignores any contents of `xsl:param`.

The use of the `expr` attribute is an extension to the XSLT specification.

- Specify *template_body*. The XSLT processor instantiates this template to obtain the default value of the parameter.
- If you do not specify the `select` attribute, the `expr` attribute, or *template_body*, the default value of the parameter is an empty string.

For any use of the `xsl:param` element, there is a region of the stylesheet tree within which the binding is visible. This region includes the siblings that follow the `xsl:param` instruction together with their descendants. Within this region, any binding of the parameter that was visible on the `xsl:param` element itself is hidden. Thus, only the innermost binding of a parameter is visible. The set of parameter bindings in scope for an expression consists of those bindings that are visible at the point in the stylesheet where the expression occurs.

The `xsl:param` instruction can be a top-level element. If it is, it declares a global parameter that is visible to the entire stylesheet. When the XSLT processor evaluates the `select` or `expr` attribute in a top-level `xsl:param` instruction, the current node is the root node of the document.

Passing parameters to templates

Use the `xsl:with-param` instruction to pass a value for a parameter to a template. See [“xsl:with-param”](#) on page 447.

xsl:preserve-space

The `xsl:preserve-space` instruction is not supported by Stylus Studio. If this instruction is in a stylesheet, it is ignored.

xsl:processing-instruction

Adds a processing instruction node to the result tree.

Format

```
<xsl:processing-instruction name = "pi_name">  
  processing_instruction  
</xsl:processing-instruction>
```

Description

The XSLT processor interprets the `name` attribute as an attribute value template, and uses the resulting string as the target of the created processing instruction. The XSLT processor then instantiates the contents of `xsl:processing-instruction` to generate the remaining contents of the processing instruction.

Errors are reported under the following conditions:

- If the string that results from evaluating the `name` attribute is not both an `NCName` and a `PITarget` (see the XSLT Recommendation). Also, the value of the `name` attribute cannot be `xml`.
- If instantiation of the contents of the `xsl:processing-instruction` element creates anything other than characters or if the resulting string contains the substring `"?>`.

Example

```
<xsl:processing-instruction name = "xml-stylesheet">  
  href="book.css" type="text/css"  
</xsl:processing-instruction>
```

This instruction creates the following processing instruction in the result document:

```
<?xml-stylesheet href="book.css" type="text/css"?>
```

xsl:sequence

Used within a sequence constructor to construct a sequence of nodes or atomic values. The sequence is returned as a result of the instruction.

Format

```
<xsl:sequence
  select = expression>
  [xsl:fallback]
</xsl:sequence>
```

Description

Unlike most other instructions, `xsl:sequence` can return a sequence containing existing nodes, rather than constructing new nodes. The items comprising the result sequence are selected using the `select` attribute. When `xsl:sequence` is used to add atomic values to a sequence, the effect is very similar to the `xsl:copy-of` instruction.

Any optional `xsl:fallback` instructions are ignored by XSLT 2.0 processors, but they can be included to define fallback behavior for XSLT 1.0 processors.

Example

This code produces the output, 37.

```
<xsl:variable name="values" as="xs:integer*">
  <xsl:sequence select="(1,2,3,4)"/>
  <xsl:sequence select="(8,9,10)"/>
</xsl:variable>
<xsl:value-of select="sum($values)"/>
```

xsl:sort

Sorts the set of nodes selected by an `xsl:apply-templates` or `xsl:for-each` instruction.

Format

```
<xsl:sort
  [select="expression" | expr="expression"]
  [optional_attribute]/>
```

Description

The `xs1:sort` instruction must be the child of an `xs1:apply-templates` or `xs1:for-each` instruction. Each `xs1:apply-templates` and `xs1:for-each` instruction can contain more than one `xs1:sort` instruction. The first `xs1:sort` child specifies the primary sort key. The second `xs1:sort` child, if any, specifies the secondary sort key, and so on.

When an `xs1:apply-templates` or `xs1:for-each` element contains an `xs1:sort` instruction, the selected nodes are processed in the order specified by the `xs1:sort` instructions. When `xs1:sort` elements are in an `xs1:for-each` element, they must appear first before all other child elements.

You can specify the sort key by using the `select` attribute, whose value is an expression. For each node selected by the `xs1:apply-templates` or `xs1:for-each` instruction, the XSLT processor evaluates the expression using the node as the context node. The resulting string is the sort key for that node. If you do not specify the `select` attribute, the XSLT processor uses the string value of the node as the sort key.

When all sort keys for two nodes are equal, nodes remain in document order.

The following optional attributes on `xs1:sort` determine how the XSLT processor sorts the list of sort keys. The XSLT processor interprets each of these attribute values as an attribute value template.

- `data-type` specifies the data type of the strings. The following values are allowed:
 - `text` specifies that the sort keys should be sorted lexicographically. All text sorting is based on Unicode text values.
 - `number` specifies that the sort keys should be converted to numbers and then sorted according to the numeric value. Sort keys that are strings that do not match the syntax for numbers are sorted as zeros.

The default value is `text`.

- `order` specifies whether the strings should be sorted in ascending or descending order. The default is ascending. If the value of the `data-type` attribute is `text`, ascending means that keys are sorted in alphabetical order, and descending means that keys are sorted in reverse alphabetical order. If the value of `data-type` is `number`, ascending means that keys are sorted in increasing order, and descending means that keys are ordered in descending order.

The XSLT processor can evaluate `xs1:sort order` at run time by using an attribute value template. For example:

```
<xs1:sort order="{ $order }"
```

`$order` is a run-time specified attribute value template.

The XSLT processor ignores the `lang` and `case-order` attributes.

Example

The following example is from the W3C XSLT Recommendation. Suppose an employee database has the following form:

```
<employees>
  <employee>
    <name>
      <first>James</first>
      <last>Clark</last>
    </name>
    ...
  </employee>
</employees>
```

The following stylesheet fragment sorts the list of employees by name:

```
<xsl:template match="employees">
  <ul>
    <xsl:apply-templates select="employee">
      <xsl:sort select="name/last"/>
      <xsl:sort select="name/first"/>
    </xsl:apply-templates>
  </ul>
</xsl:template>
<xsl:template match="employee">
  <li>
    <xsl:value-of select="name/first"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="name/last"/>
  </li>
</xsl:template>
```

xsl:strip-space

The `xsl:strip-space` instruction is not supported by Stylus Studio. If this instruction is in a stylesheet, it is ignored.

xsl:stylesheet

Specifies the start of a stylesheet.

Format

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0" >
  stylesheet_body
</xsl:stylesheet>
```

Description

A stylesheet must specify the `xsl:stylesheet` element unless it contains only a literal result element as the root element. The `xsl:transform` instruction is a synonym for `xsl:stylesheet`.

All XSLT elements must appear between the `<xsl:stylesheet>` and `</xsl:stylesheet>` tags. An element that is a child of an `xsl:stylesheet` element is a top-level element.

xsl:template

Specifies a template rule.

Format

```
<xsl:template
  [match = "pattern"]
  [name = "qname"]
  [mode = "mode"]
  [priority = "priority"]>
  template_body
</xsl:template>
```

Description

The `match` attribute is required except when you specify the `name` attribute. The pattern you specify for the `match` attribute identifies the source node or set of source nodes to which the template rule applies.

The optional `name` attribute specifies a name for the template. You can use the name of a template to invoke it with the `xsl:call-template` instruction. The value you specify for `name` must be a qualified name. If you specify a `name` attribute, a `match` attribute is not required.

The optional `mode` attribute prevents the template from matching nodes selected by an `xs1:apply-templates` instruction that specifies a different mode. The value of `mode` must be a qualified name or an asterisk (*). If you specify an asterisk, it means match any node.

If an `xs1:apply-templates` instruction contains a `mode` attribute, the `xs1:apply-templates` instruction can apply to only those `xs1:template` instructions that specify a `mode` attribute with the same value. If an `xs1:apply-templates` instruction does not contain a `mode` attribute, the `xs1:apply-templates` instruction can apply to only those `xs1:template` instructions that do not specify a `mode` attribute.

If you specify the `match` and `mode` attributes, they have no effect if the template is instantiated by the `xs1:call-template` instruction. If you specify the `name` attribute, you can still instantiate the template as a result of an `xs1:apply-templates` instruction.

If two or more templates have the same name, Stylus Studio uses the template that appears last in the stylesheet.

The template body contains literal results and XSLT instructions. The XSLT processor instantiates the template body for each node identified by *pattern*. This means the XSLT processor copies literal results to the result document and executes the XSLT instructions.

If there is more than one matching template rule, the XSLT processor chooses the matching template rule with the higher priority. If both have the same priority, the XSLT processor chooses the one that occurs last in the stylesheet.

For examples and additional information about templates, see [“Working with Templates”](#) on page 382.

Tip You can create an `xs1:template` element automatically using the XSLT mapper.

xsl:text

Adds a text node to the result tree.

Format

```
<xsl:text [disable-output-escaping="yes|no"]>
  text_node_contents
</xsl:text>
```

Description

The XSLT processor reports an error if instantiating *text_node_contents* results in anything other than characters.

You can also add text nodes to result documents by embedding the text in elements that you define.

You can specify the `disable-output-escaping` attribute of the `xsl:text` instruction. The allowed values are `yes` or `no`. The default is `no`. If the value is `yes`, the text node generated by instantiating the `xsl:text` element is output without any escaping. For example:

```
<xsl:text disable-output-escaping="yes">&lt;</xsl:text>
```

This instruction generates the single character `<`.

Examples

The following fragment adds two text nodes by embedding text.

```
<xsl:template match = "/">
  <html>
    <head><title>Authors and Their Books</title></head>
    <body>
      <intro>Books in stock are listed here.</intro>
      ...
    </body>
  </html>
</xsl:template>
```

The next example specifies the `xsl:text` instruction:

```
<xsl:text>Following is a list of authors.</xsl:text>
```

xsl:transform

The `xsl:transform` instruction is a synonym for `xsl:stylesheet`. See “[xsl:stylesheet](#)” on page 442.

xsl:value-of

Creates a new text node that contains the string value of an expression.

Format

```
<xsl:value-of select="expression"  
[disable-output-escaping="yes|no"]/>
```

Description

The XSLT processor evaluates *expression* and converts the result to a string. If the string is not empty, a text node is created and added to the result. If the string is empty, the `xsl:value-of` instruction has no effect.

You can specify the `disable-output-escaping` attribute of the `xsl:value-of` instruction. The allowed values are `yes` and `no`. The default is `no`. If the value is `yes`, the text node generated by instantiating the `xsl:value-of` element is output without any escaping.

Tip You can create an `xsl:value-of` element automatically using the XSLT mapper.

Example

```
<xsl:template match = "author">  
  <p>  
    <xsl:value-of select = "first-name"/>  
    <xsl:text> </xsl:text>  
    <xsl:value-of select = "last-name"/>  
  </p>  
</xsl:template>
```

This example creates an HTML paragraph from an `author` element. The `author` element has `first-name` and `last-name` children. The resulting paragraph contains the value of the first `first-name` child element of the current node, followed by a space, followed by the value of the first `last-name` child element of the current node.

xsl:variable

Declares a variable and binds a value to that variable.

Format

```
<xsl:variable name="variable_name"  
  [select = "expression2"]  
  [expr = "expression3"]>  
  template_body  
</xsl:variable>
```

Description

The name attribute is required, and it must be a string. The value of the name attribute is a qualified name.

The value that you bind to a variable can be an object of any of the types that are returned by expressions. You can specify the value of the variable in several ways:

- Specify the `select` attribute. The value of the `select` attribute must be an expression. The XSLT processor evaluates the expression, and the result is the value of the variable. If you specify the `select` attribute, you must not specify any contents for the `xsl:variable` instruction. In other words, do not specify `template_body`.
- Specify the `expr` attribute. It is interpreted as an attribute value template. It allows computation of the value expression.

The `expr` attribute of the `xsl:variable` instruction is an extension of the XSLT standard. If you want to use an XSLT processor other than the Stylus Studio processor, you cannot specify the `expr` attribute in your stylesheet.

- Specify `template_body`. The XSLT processor instantiates this template to obtain the value of the variable. If you specify `template_body`, you must not specify the `select` attribute.
- Specify none of the above. In this case, the value of the variable is an empty string.

The difference between the `xsl:param` and `xsl:variable` instructions is that `xsl:param` defines a default value while `xsl:variable` defines a fixed value.

For any use of the `xsl:variable` element, there is a region of the stylesheet tree within which the binding is visible. This region includes the siblings that follow the `xsl:variable` instruction together with their descendants. Within this region, any binding of the variable that is visible on the `xsl:variable` element itself is hidden. Thus, only the innermost binding of a variable is visible. The set of variable bindings in scope for an

expression consists of those bindings that are visible at the point in the stylesheet where the expression occurs.

The `xsl:variable` instruction can be a top-level element. If it is, it declares a global variable that is visible to the entire stylesheet. When the XSLT processor evaluates the `select` or `expr` attribute in a top-level `xsl:variable` instruction, the current node is the root node of the document. The `xsl:variable` instruction is also allowed anywhere in a template that an XSLT instruction is allowed.

xsl:when

See “[xsl:choose](#)” on page 417.

xsl:with-param

Passes a parameter value to a template.

Format

```
<xsl:with-param name = "parameter_name"  
  [select = "expression1"]  
</xsl:with-param>
```

Description

The `xsl:with-param` instruction passes a parameter value to a template. If the template has no matching `xsl:param` declaration, the XSLT processor ignores the parameter. The value of *parameter_name* is a qualified name.

The name attribute is required, and it must be a string. The value of the name attribute is a qualified name.

The value that you pass to a template can be an object of any of the types that are returned by expressions. You can specify the value of the parameter in several ways:

- Specify the `select` attribute. The value of the `select` attribute must be an expression. The XSLT processor evaluates the expression, and the result is the value of the parameter. If you specify the `select` attribute, you must not specify any contents for the `xsl:with-param` instruction. In other words, do not specify *parameter_value*.
- Specify the `expr` attribute. It is interpreted as an attribute value template. It allows computation of the value expression.

- Specify *parameter_value*. If you specify *parameter_value*, you must not specify the `select` or `expr` attribute.
- Specify none of the above. In this case, the value of the parameter is an empty string.

The `xs1:with-param` element must be a child of `xs1:apply-templates` or `xs1:call-template`.

You can specify the `xs1:with-param` instruction in `xs1:call-template` and `xs1:apply-template` instructions.

Example

Suppose you specify the following parameter for a template:

```
<xs1:template name="Appendix">
  <xs1:param name = "heading"> 1. </xs1-param>
  . . .
</xs1:template>
```

You can pass another value for this variable as follows:

```
<xs1:call-template name = "Appendix">
  <xs1:with-param name = "heading"> A. </xs1:with-param>
</xs1:call-template>
```

Chapter 5 **Creating XSLT Using the XSLT Mapper**



The XSLT Mapper is available only in Stylus Studio XML Enterprise Suite and Stylus Studio XML Professional Suite.

In addition to writing XSLT manually in the XSLT text editor, Stylus Studio provides a graphical tool, the XSLT mapper, that allows you quickly compose XSLT without writing any code. This chapter describes the XSLT mapper, how to use it, and its relationship to the XSLT displayed on the **XSLT Source** tab.

For a brief introduction to the mechanics of using the XSLT mapper and some of its features, see [“Using the XSLT Mapper – Getting Started”](#) on page 47.

This chapter covers the following topics:

- [“Overview of the XSLT Mapper”](#) on page 450
- [“Source Documents”](#) on page 458
- [“Target Structures”](#) on page 465
- [“Mapping Source and Target Document Nodes”](#) on page 468
- [“Working with XSLT Instructions in XSLT Mapper”](#) on page 470
- [“Processing Source Nodes”](#) on page 476
- [“Creating and Working with Templates”](#) on page 483
- [“Creating an XSLT Scenario”](#) on page 485

Overview of the XSLT Mapper

The XSLT mapper helps you compose XSLT that aggregates data from one or more source documents, regardless of their origin or XML. For example, an inventory application might use information from multiple vendors, each of whom organizes invoices in a different way. You can use the XSLT mapper to identify source documents, map the relevant nodes from each to a target document, and in doing that define any required XSLT instructions, XPath or Java functions, and logical operators graphically.

To use the XSLT mapper to create an XSLT stylesheet, you start by specifying one or more source documents and one target document.

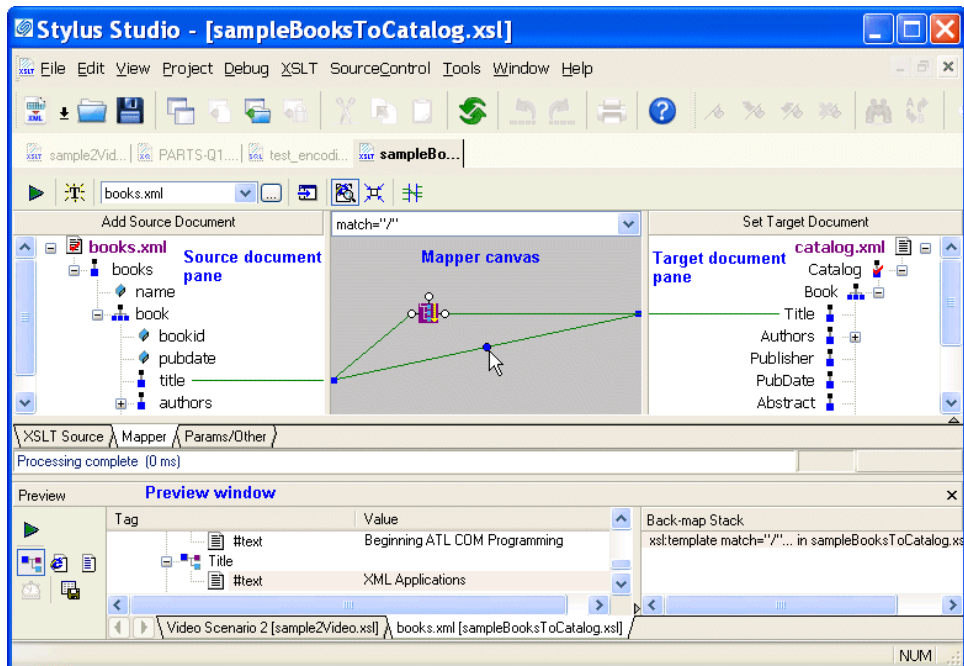


Figure 216. Example of XSLT Mapper

The **Mapper** tab consists of these areas:

- Source document pane, in which you add one or more source documents.
- Target structure pane, in which you specify the structure of the result you want the XSLT to return.
- Mapper canvas, on which you can define conditions, functions, and operations for source document nodes to filter return values that are then mapped to the target node.

- Source code pane (not shown in [Figure 216](#)). The source code pane allows you to view the source code while using the mapper. This is a great way to see how changes to the mapper affect the source, without the need to switch to the **XSLT Source** tab. Of course, the **XSLT Source** tab is available if you prefer working with the source using a full-page view. All views – Mapper tab, XSLT Source tab, and the source pane – are synchronized. When displayed, the source pane spans the width of the XSLT editor.

As you link elements and define XSLT instruction and function blocks in the mapper, Stylus Studio composes XSLT for you, which is visible (and editable) any time you click the XSLT editor's **XSLT Source** tab. When you have finished mapping, you can apply the stylesheet to XML documents that have the same schema as the source document. The result document also has the same schema as the destination document.

As with the **XSLT Source** tab, you can preview XSLT results from the **Mapper** tab by clicking the **Preview Result** button (▶). Debugging, however, can be performed from the **XSLT Source** tab only.

This section covers the following topics:

- [Example](#) on page 451
- [Graphical Support for Common XSLT Instructions and Expressions](#) on page 452
- [Setting Options for the XSLT Mapper](#) on page 453
- [Simplifying the Mapper Canvas Display](#) on page 454
- [Exporting Mappings](#) on page 456
- [Searching Document Panes](#) on page 457
- [Ensuring That Stylesheets Output Valid XML](#) on page 457

Example

Suppose you open the XML mapper and select `books.xml` as the source document and `catalog.xml` as the target document. You then map elements in the `books.xml` document or structure to elements in the `catalog.xml` document or structure. The result is a stylesheet that you can apply to `books.xml` and to other files that have a structure similar to that of `books.xml`. When you apply this stylesheet, the result is an XML document whose structure is consistent with that of `catalog.xml`.

Now suppose you want to apply a stylesheet to `catalog.xml` and output an XML file that has a structure similar to `books.xml`. To do this, you must use the XSLT mapper to create a second stylesheet. This time, `catalog.xml` is the source document and `books.xml` is the

destination document. The result of this mapping is a stylesheet that you can apply to documents that have a structure similar to that of `catalog.xml`.

Graphical Support for Common XSLT Instructions and Expressions

The XSLT mapper has graphical support for

- XSLT instructions
- XPath functions
- Logical operators
- Java Functions

Using special symbols, called *blocks*, you can quickly and easily create complex XSLT without writing any code, as shown in [Figure 217](#):

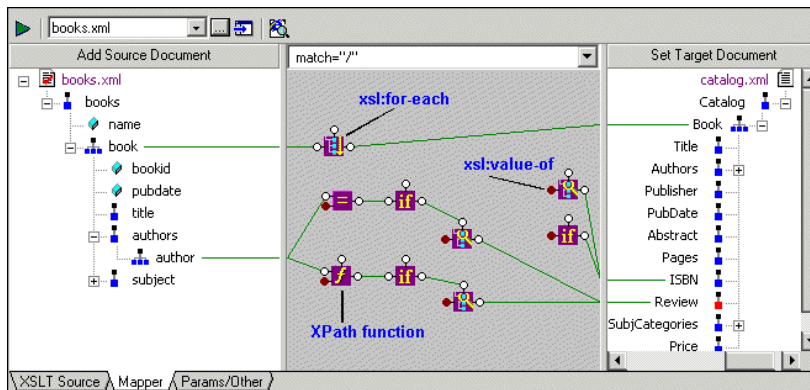


Figure 217. XSLT Operation, Function, and Logical Operator Blocks

Blocks can be created

- Automatically, when you link one node to another. For example, if you link repeating elements in the source and target documents, Stylus Studio automatically creates an `xs1:for-each` instruction block in the mapper.
- Manually, by selecting the instruction or expression you want to create from the shortcut menu on the mapper canvas (right click on the mapper canvas to display this menu).
- By reverse-engineering the XSLT that you write on the **XSLT Source** tab – when you click the **Mapper** tab, XSLT that can be represented graphically is displayed on the mapper canvas.

See “[Working with XSLT Instructions in XSLT Mapper](#)” on page 470 and “[Processing Source Nodes](#)” on page 476 to learn more about working with blocks in the XSLT mapper.

Setting Options for the XSLT Mapper

There are a few options you can set that affect the XSLT stylesheets generated by the XSLT mapper. To display the **Options** dialog box, in the Stylus Studio tool bar, select **Tools > Options**.

Under **Module Settings > XSLT Editor**, click **Mapper**. The mapper has an option that determines whether Stylus Studio creates empty elements for unlinked nodes when the associated schema specifies that the elements are required. You might want to select this option to help ensure that your XSLT generates valid XML by ensuring that all required elements are accounted for.

Tip A red check appears on the symbol for required nodes in the target document tree displayed in the structure pane.

Among other options under the **XSLT Editor** heading, consider clicking **XSLT Settings** and specifying whether or not you want Stylus Studio to save scenario metainformation in the stylesheet. Scenario metainformation includes anything specified in the **Scenario Properties** dialog box – source and output URLs, parameter values, post-processing options, and so on.

Note If you select this option, Stylus Studio also saves mapper metainformation in the stylesheet; mapper metainformation includes the names of source files, node mapping information, and so on.

If you choose not to save scenario metainformation in the stylesheet, and if the stylesheet belongs to a project, Stylus Studio saves mapper metainformation in the project. If the stylesheet does not belong to a project, and you choose not to save metainformation in the stylesheet, mapping metainformation is not saved.

Simplifying the Mapper Canvas Display

By default, the XSLT Mapper displays all links between source and target document nodes, regardless of whether or not the node associated with a link is currently visible in the **Source Document** or **Target Document** pane. Further, as your XSLT code becomes more complex, the mapper canvas can become dense with graphical representations of the functions defined in the code and the links that represent them. Consider this example of `sample2Video.xsl`, one of the sample XSLT stylesheets in the Examples project installed with Stylus Studio.

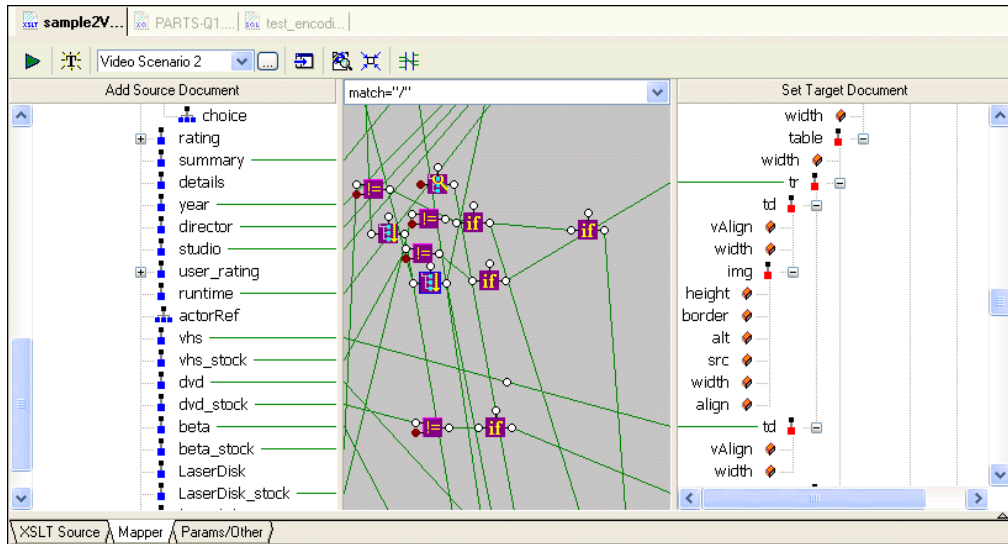


Figure 218. Mapper Shows Links to All Nodes, Visible or Not

You can hide links for nodes that are not currently visible in the **Source Document** or **Target Document** pane by clicking the **Hide Links for Nodes that are not Visible** button, as shown in [Figure 219](#):

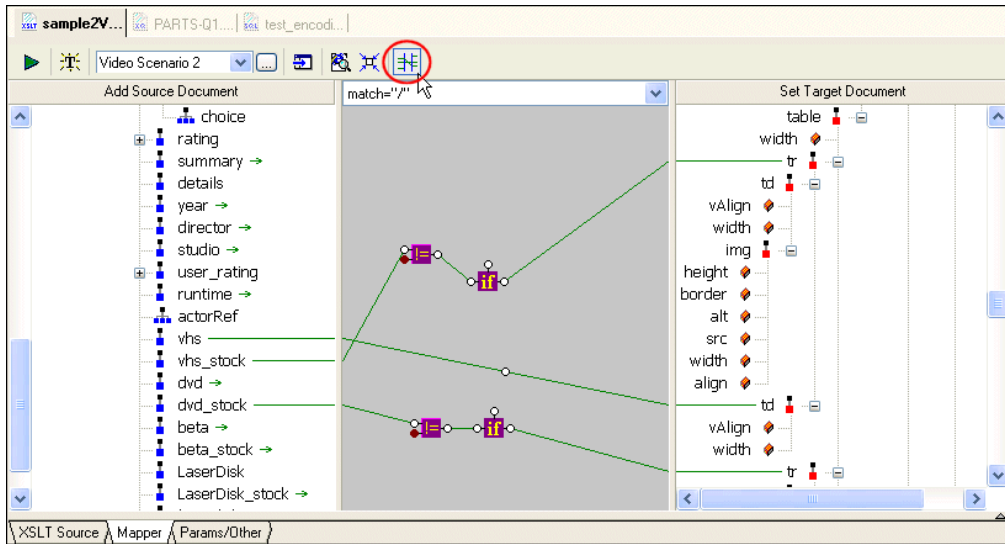


Figure 219. Simply the Mapper by Hiding Links

When you use this feature, Stylus Studio displays

- Links in the Mapper canvas only if both nodes are currently visible in the document panes
- Green arrows (like the ones shown in [Figure 220](#)) in the document panes if only one of two linked nodes is currently visible.

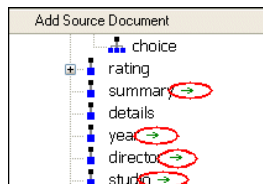


Figure 220. Arrows Identify Partially Available Links

Other Mapper Display Features

In addition to displaying links for only those nodes that are visible in both document panes, you can use the document node shortcut menu (right-click on a node in a document pane) to

- Show links to a specific node
- Hide links to a specific node
- Show/hide all links

Exporting Mappings

You can export a mapping – source and target document trees and Mapper canvas contents – as an image file. The default image format is JPEG (.jpg), but you can choose from other popular image file formats such as .bmp and .tiff.

The exported image reflects the document trees at the time you export the image – if you have collapsed a node in Stylus Studio, for example, that node is also collapsed in the exported image. However, the exported image includes the entire document tree and Mapper canvas, not just what is currently visible on the **Mapper** tab.

By default, all source-target document links are displayed. However, if you have chosen to [hide or show links](#) for only certain nodes, the exported image reflects that choice and displays only the links for the nodes as you have specified. See “[Simplifying the Mapper Canvas Display](#)” on page 454 for more information on hiding and showing links.

◆ To export an XSLT mapping:

1. Optionally, hide links for any nodes in the source or target documents that you do not want to appear in the exported image.
2. Select **XSLT > Export Mapping as Image** from the Stylus Studio menu. Stylus Studio displays the **Save As** dialog box.
3. Specify a URL for the file.
4. Optionally, change the image type. (The default is JPEG; .bmp and .tiff are also available.)
5. Click **Save**.

Searching Document Panes

You can search document panes using the **Find** dialog box.

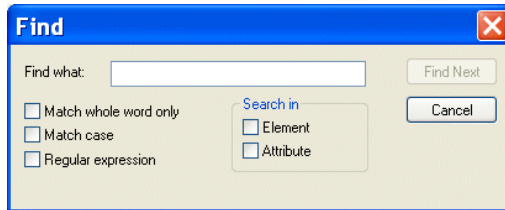


Figure 221. You Can Search Document Panes

You can restrict your search to elements and/or attributes, and you can even search using regular expressions to define your match pattern.

◆ **To display the Find dialog box:**

1. Right-click in the document pane.
2. Select **Find** from the shortcut menu.

Ensuring That Stylesheets Output Valid XML

Stylus Studio cannot automatically generate a stylesheet that will always generate a valid XML document. As defined by the W3C, an XML document is considered to be valid if it conforms to the DTD with which it is associated.

For example, consider a stylesheet that has required attributes. In order to specify meaningful values for them, you need to have insight as to the semantics of the operation the stylesheet is performing, and it is difficult for any application to infer this type of information. Always check to see that the stylesheets you create using the XSLT mapper generate valid input.

Steps for Mapping XML to XML

◆ **To create an XSLT stylesheet using the XSLT mapper:**

1. From the Stylus Studio menu bar, select **File > New > XSLT: Mapper**. Stylus Studio displays the XSLT editor with the **Mapper** tab selected.
2. Select one or more source documents and a target document.

3. In both the source and the target panes, click the root element and then press the asterisk key (*) in the numeric key pad to expand the schema tree.
4. Map nodes in the source documents to nodes in the target document, define XSLT instructions and functions, and create named and matched templates using the mapper's graphical tools.

Tip

Consider working through the source document in document order as you map source and target elements.

5. Check the **XSLT Source** view from time to time. This allows you to confirm that the stylesheet is doing what you expect it to do (and it is also a good way to teach yourself XSLT). Changes you make directly to the source are reflected on the **Mapper** tab, and vice versa.

Each of these steps is described in greater detail in the following sections.

Source Documents

In Stylus Studio, a source document in the XSLT mapper can be an XML document, an XML Schema (XSD), or a document type definition (DTD). The role of a source document is to provide Stylus Studio with a structure that it can use to compose the XSLT stylesheet, based on how you map individual source document elements and attributes to nodes in the target structure. Stylus Studio infers the target structure from the document (XML, XSD, or DTD) you specify and displays this structure on the **Mapper** tab.


This section covers the following topics:

- [“Choosing Source Documents”](#) on page 458
- [“Source Documents and XML Instances”](#) on page 459
- [“How to Add a Source Document”](#) on page 462
- [“How to Remove a Source Document”](#) on page 463
- [“How Source Documents are Displayed”](#) on page 463

Choosing Source Documents

You can use one or more source documents to build a stylesheet in the Stylus Studio XSLT mapper. You might want to select more than one document if you need their elements or attributes to fully describe the target structure or the desired XSLT result content, or if you want to aggregate multiple sources in a single document, for example.

If you choose an XSD or DTD document, you must also choose an XML instance document to associate with it. Stylus Studio uses the instance document associated with an XSD or DTD source document to generate the XPath document() function in the finished XSLT. As a result, it is this document that is used to preview XSLT results.

Tip If you want to examine the contents of the XML document specified as the source file in the scenario, click **Open XML From Scenario** , which is at the top of the XML mapper window. Stylus Studio displays the source document in the XML editor.

See “[Source Documents and XML Instances](#)” on page 459 to learn more about how Stylus Studio treats source documents. See “[Creating an XSLT Scenario](#)” on page 485 to learn more about XSLT scenarios.

Source Documents and XML Instances

As described previously, Stylus Studio uses the source documents you specify to display a structure you can use to create mappings to the target structure. In addition to the document structure, Stylus Studio needs document content information in order to compose a correct XSLT stylesheet. You provide this information by associating a XML instance to each source document you specify.

Types of associations

Source documents can have one of three associations, each of which has implications for the XPath expressions written by Stylus Studio, which uses these documents when it composes the XSLT stylesheet. A source document can be associated with

- Itself. That is, the document represented by structure displayed on the **Mapper** tab and the XML instance are one in the same. In this situation, Stylus Studio generates the document() function in the XSLT stylesheet. For example:

```
document("file://c:\Program Files\Stylus Studio\examples\simpleMappings\catalog.xml")/books/book
```

Note The previous example shows the XSLT that results when an XML document is used to specify the source structure. This is not possible with XSD or DTD source documents.

- The XML document specified in the XSLT scenario. Only one source document can be associated with the XSLT scenario. In this situation, Stylus Studio does not generate the document() function in the XSLT stylesheet. In this situation, the

Creating XSLT Using the XSLT Mapper

document() function is not necessary because Stylus Studio uses the XSLT input document specified in the **Scenario Properties** dialog box.

By default, Stylus Studio uses the first XML document you add to the XSLT mapper as the source XML for the XSLT scenario, as shown here:

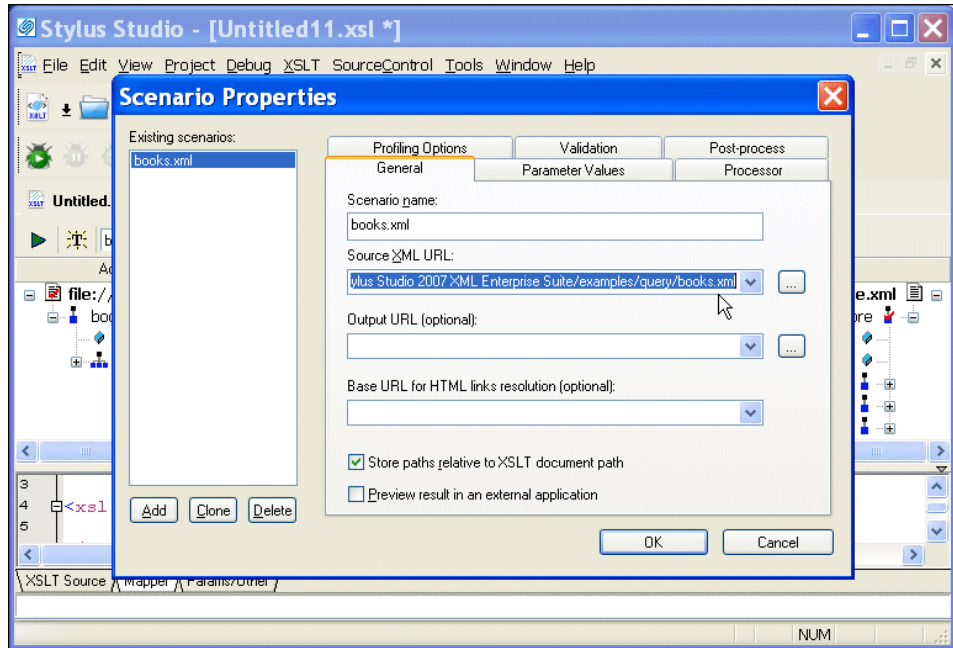


Figure 222. XSLT Scenario

The document specified in the **Source XML URL** field on the **Scenario Properties** dialog box is the document to which the XSLT is applied when you preview the XSLT. You can select this association for another XML document if you choose, but only one source document may have this association.

Note




If you specify an XML document as the first source document, Stylus Studio creates a scenario for you automatically, using that document as the scenario's source XML. If you specify some other type of document (XSD or DTD), Stylus Studio prompts you to create a scenario – and to specify an XML document as the source – when you preview the XSLT. See [“Creating an XSLT Scenario”](#) on page 485.

- Some other XML instance. A XSD or DTD document used as a mapper source document must always be associated with an XML instance. In this situation, Stylus Studio generates the `document()` function in the XSLT stylesheet when accessing nodes of the document structure.

Source document icons

Stylus Studio uses different document icons to indicate how a source document structure is related to corresponding XML document content.

Table 54. Source Document Icons

<i>Icon</i>	<i>Meaning</i>
	The source document is associated with itself. This is the default for most XML documents (and XML documents only).
	The source document is associated with default XML document specified in the Source XML URL field in the XSLT scenario. This is the case with the first XML document you add to XSLT mapper, but you can change this association manually if you choose. See “How to change a source document association” on page 461.
	The source document is associated with a separate XML document instance. XSD and DTD source documents are always associated with an XML instance.

How to change a source document association

◆ To change a source document association:

1. Click the **Mapper** tab if necessary.
2. Right click the source document whose association you want to change. The source document shortcut menu appears.
3. Click **Associate With**, and then select the document you want to associate with the source document.

How to Add a Source Document

◆ **To add a source document to XSLT mapper:**

1. Click the **Mapper** tab if necessary.
2. Click the **Add Source Document** button at the top left of the **Mapper** tab.
The **Open** dialog box appears.
3. Select the document you want to use as the source document to map to the target document.
4. Click **Open**.

If you selected an XML document in [Step 3](#), the document appears in the source document pane of the **Mapper** tab. Go to [Step 5](#).

If you selected an XSD or DTD document, Stylus Studio displays the **Choose Root Element** dialog box.

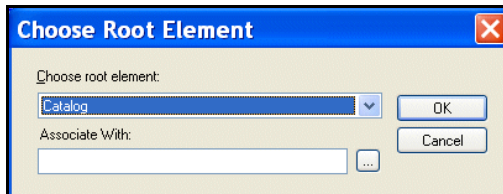


Figure 223. Choose Root Element Dialog Box

You use the **Associate With** field to associate the XSD or DTD with an XML instance.

Note

The **Associate With** field appears only when you add a second document to the XSLT mapper source and that document is an XSD or DTD. You use it to specify the XML instance that you want to associate with the XSD or DTD. This field does not appear if the XSD or DTD is the first source document you add to the XSLT mapper – Stylus Studio uses the XML Source document specified in the **Scenario Properties** dialog box as the XML instance in this case.

- a. Select the element from the XSD or DTD document that you want to use as the root element. The **Choose root element** drop-down list displays elements defined in the document you selected in [Step 3](#).
- b. Use the **Browse** () button to specify the XML instance to which you want to associate the document you have chosen as your structure. The root element of the XML document you select should be the same as the element you selected as the root element from the XSD or DTD document.

- c. Click **OK**.
The document appears in the source document pane of the **Mapper** tab. Go to [Step 5](#).
5. To add another source document, return to [Step 2](#).

How to Remove a Source Document

Note A source document cannot be removed from XSLT mapper if it is mapped to the target structure. See [“Removing Source-Target Maps”](#) on page 470.

◆ **To remove a source document from the XSLT mapper:**

1. Click the **Mapper** tab if necessary.
2. Remove any maps from the source document to the target schema. (See [“Removing Source-Target Maps”](#) on page 470 if you need help with this step.)
3. Right click on the source document.
The source document shortcut menu appears.
4. Select **Remove Schema**.

How Source Documents are Displayed

A source document is represented in the mapper using a document icon; its name is displayed using a different color to help distinguish the document from elements and attributes. The document icon is modified based on the source document’s association with other documents. See [“Source Documents and XML Instances”](#) on page 459 for more information on this topic.

By default, only the file name itself is displayed; if you want, you can display the document's full path by selecting **Show Full Path** on the document's shortcut menu. (Right-click on the document name to display the shortcut menu.)

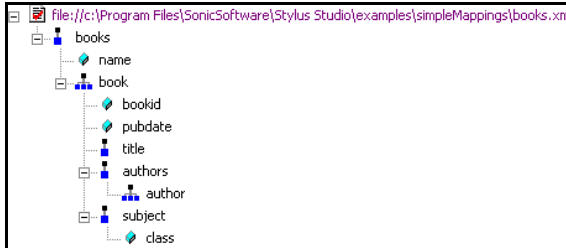





Figure 224. Source Document Display

Source documents are displayed using the tree view; you can use your standard keyboard's *, +, and - number pad keys to expand and collapse selected documents.

Document structure symbols

Stylus Studio uses the following symbols to represent nodes in both source and target document structures

Table 55. Document Structure Symbols

<i>Symbol</i>	<i>Meaning</i>
	Repeating element
	Element
	Attribute

See “[Source document icons](#)” on page 461 to learn about the different ways source document icons are depicted.

Getting source document details

If you want details about the source document that are not available in tree view, you can open the document by selecting **Open** from the document's shortcut menu. When you open a document this way, Stylus Studio displays it in the XML editor. XSD and DTD documents are displayed on the XML editor's **Schema** tab.

Target Structures

There are two ways to specify an XSLT target structure:

- You can select an existing document from which Stylus Studio infers a structure and, optionally, modify the structure. Existing nodes in a target structure are displayed in blue. Nodes that you add are displayed in red.
- You can build a structure from scratch, starting with the root element and defining other elements and attributes as needed. Nodes for target structures you define are displayed in red.

This section covers the following topics:

- [“Using an Existing Document”](#) on page 465
- [“Building a Target Structure”](#) on page 465
- [“Modifying the Target Structure”](#) on page 467

Using an Existing Document

◆ **To use an existing document to provide the XSLT target structure:**

1. Click the **Mapper** tab if necessary.
2. Click the **Set Target Document** button at the top left of the **Mapper** tab.
The **Open** dialog box appears.
3. Select the document you want to use to provide the target structure for defining the mapping (XML, XSD, or DTD).
4. Click **Open**.
The structure of the document you select appears in the target document pane of the **Mapper** tab.

Building a Target Structure

To build a target structure from scratch, you first create a root element, and then define child elements and attributes as needed.

How to create a root element

◆ **To create a root element:**

1. Click the **Mapper** tab if necessary.
2. Right click the area underneath the **Set Target Document** button.
The target document shortcut menu appears.
3. Select **Create Root Element**.
The **Name** dialog box appears.

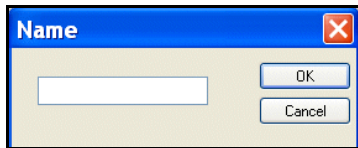


Figure 225. Name Dialog Box

4. Type a name for the root element and click **OK**.
The root element you specified appears in the target document pane of the **Mapper** tab.

How to create elements and attributes

You can create elements and attributes in a new or existing target structure.

◆ **To create elements and attributes:**

1. Click the **Mapper** tab if necessary.
2. Select the attribute or element to which you want to add a child element or attribute.
If you have just created a root element, select the root element.
3. Right click the area underneath the **Set Target Document** button.
The target document shortcut menu appears.
4. Choose one of the following:
 - **Add Attribute**
 - **Add Child Element**
 - **Insert Element After** (This choice is not applicable to the root element; it creates the element as a sibling of the selected element.)

The **Name** dialog box appears.

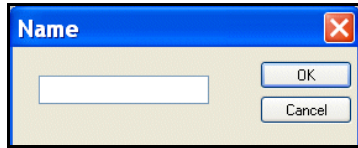


Figure 226. Name Dialog Box

5. Type a name for the node and click **OK**.
The node you specified is added to the target structure in the **Mapper** tab.

Modifying the Target Structure

This section describes the techniques you can use to modify the structure and content of an XSLT mapper target structure. It covers the following topics:

- [“Adding a Node”](#) on page 467
- [“Removing a Node”](#) on page 467

Adding a Node

See [“How to create elements and attributes”](#) on page 466.

Removing a Node

Note Before you can remove a node, you must delete any links to that node. See [“Removing Source-Target Maps”](#) on page 470.

◆ To remove a node from the target structure:

1. Remove any links to the node you want to remove from the target structure. See [“Removing Source-Target Maps”](#) on page 470 if you need help with this step.
2. Select the node and press the Delete key.
Alternative: Right-click the node and select **Remove Node** from the shortcut menu.

Mapping Source and Target Document Nodes

You map a source document node to a target structure node using drag and drop to create a link between the two nodes. Stylus Studio composes XSLT based on these maps.


This section covers the following topics:

- [“Preserving Mapper Layout”](#) on page 468
- [“Left and Right Mouse Buttons Explained”](#) on page 468
- [“How to Map Nodes”](#) on page 469
- [“Removing Source-Target Maps”](#) on page 470

Tip You can also map source document nodes to XSLT instruction blocks, XPath and Java function blocks, and logical operators. See [“Working with XSLT Instructions in XSLT Mapper”](#) on page 470 and [“Processing Source Nodes”](#) on page 476.

Preserving Mapper Layout

As you add function blocks to the XSLT mapper, Stylus Studio places them in the center of the mapper canvas. You can change the default placement of function blocks by dragging and drag and dropping them where you like. Stylus Studio preserves the placement you select within and across sessions (as you toggle between the mapper and the **XSLT Source** tab, for example).

As you use the splitter in the XSLT mapper to widen the source and target document panes, the size of the mapper canvas is reduced. The **Fit in Mapper Canvas** button () , located at the top of the XSLT mapper, redraws the diagram in whatever space is currently available to the mapper canvas. This feature is also available from the mapper short-cut menu (right-click anywhere on the mapper canvas to display the short-cut menu).

Left and Right Mouse Buttons Explained

You can use either the left or the right mouse button to perform the drag and drop operation used to create source-target mappings in the XSLT mapper.

If you use the left mouse button to perform the drag operation, the link always maps the source node to the target node without making any changes to the target structure. If you

use the right mouse button, Stylus Studio displays a shortcut menu that provides you with alternatives for modifying the target structure.

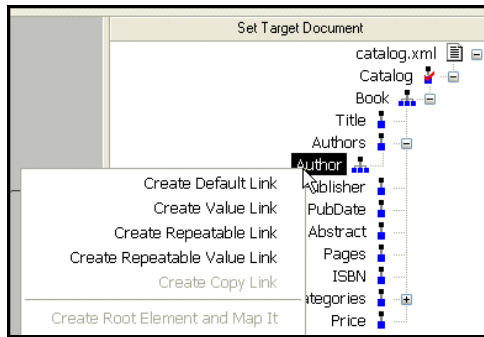


Figure 227. Shortcut Menu for Target Document Operations

Using this menu, you can

- Map a source document node to an existing target structure node – this menu choice, **Map to This Node**, is the same as creating the link using the left mouse button.
- Add a source document node (element or attribute) as an attribute of the target structure node you select and map the two nodes.
- Add a source document node as a child element of the target structure node you select and map the two nodes.
- Add a source document node as a sibling of the target structure node you select and map the two nodes.
- Copy the entire source document node – its structure and its content – to the target structure and map it.

How to Map Nodes

◆ To map nodes:

1. Using either the left or right mouse button, drag the source document element or attribute to the appropriate node on the target structure.

Tip If you need help with this step, see [“Left and Right Mouse Buttons Explained”](#) on page 468.

2. When the pointer is on the appropriate target element, release the mouse button to complete the link.

Stylus Studio draws a link between the source and target nodes you chose in [Step 1](#). If you linked two repeating elements, Stylus Studio displays a symbol representing the `xsl:for-each` instruction. See [Working with XSLT Instructions in XSLT Mapper](#) on page 470.

Removing Source-Target Maps

◆ **To remove a map from a source document node to a target element node:**

1. Select the line that represents the map you want to delete.

Tip Select the portion of the line that is drawn on the XSLT mapper canvas.

2. Press the **Delete** key.

Alternative: Select **Delete** from the line shortcut menu (right click on the line to display the shortcut menu).

Working with XSLT Instructions in XSLT Mapper

As described in [Graphical Support for Common XSLT Instructions and Expressions](#) on page 452, you can create and work with XSLT instructions in the XSLT mapper using symbols called blocks. Each supported instruction is represented by a different block (symbols distinguish one block from another), and you complete the instruction's definition graphically, using drag and drop.







This section identifies the XSLT instructions supported by the mapper, their features, and how to use them. It covers the following topics:

- [What XSLT Instructions Are Represented Graphically](#) on page 471
- [Instruction Block Ports](#) on page 471
- [Understanding Input Ports](#) on page 472
- [The Flow Port](#) on page 473
- [Adding an Instruction Block to the XSLT Mapper](#) on page 474
- [xsl:if and xsl:choose](#) on page 475

What XSLT Instructions Are Represented Graphically

The XSLT mapper represents the following XSLT instructions:

Table 56. XSLT Instruction Blocks in XSLT Mapper

<i>XSLT Instruction</i>	<i>Representation in the XSLT Mapper</i>
<code>xsl:value-of</code>	
<code>xsl:for-each</code>	
<code>xsl:if</code>	
<code>xsl:choose</code>	
<code>xsl:apply-templates</code>	
<code>xsl:call-template</code>	

Note You can create any XSLT instruction in the XSLT source, but only the instructions in [Table 56](#) are represented graphically in the XSLT mapper. XSLT instructions that are not supported by the mapper have no graphical representation.

Instruction Block Ports

All XSLT instruction blocks have at least three connectors, called *ports*. Look at the `xsl:value-of` instruction block shown in [Figure 228](#).

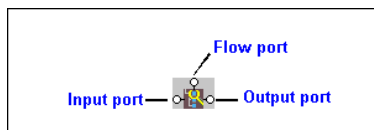


Figure 228. Example of an XSLT `xsl:value-of` Instruction Block

You use these ports to link source and target nodes, to perform processing on source document nodes, and to provide flow control as the result of a `xsl:choose` or `xsl:if`.

Ports are also part of XPath and Java function blocks, logical operator blocks, and text blocks. (See [Processing Source Nodes](#) on page 476 for information on working with these types of blocks.)

Specifying Values for Ports

After you have added an instruction block to the XSLT mapper, you need to complete its definition. You do this by linking the instruction block’s input, output, and, optionally, flow ports to nodes and other blocks in the mapper.

The way you specify values for ports varies slightly between input ports and flow and output ports, but, generally speaking, you can either

- Dragging a link from the port to a target document node or to the flow port on another instruction block.
- Double-click the port and typing a value (a string or an XPath expression, for example) in the **Value** dialog box.

Tip To see the XSLT that is being generated based on the XSLT instruction you are creating, right click the instruction and select **Go To Source** from the shortcut menu.

Understanding Input Ports

Stylus Studio interprets input ports differently for different XSLT instructions, as shown in [Table 57](#):

Table 57. XSLT Instruction Blocks in XSLT Mapper

<i>XSLT Instruction</i>	<i>Meaning of Input Port</i>
xsl:value-of	Used to define the value of the <code>select</code> attribute. For example: <code><xsl:value-of select="'Owen'"/></code>
xsl:for-each	Used to define the XPath expression for the <code>select</code> attribute. For example: <code><xsl:for-each select="books/book"></code>
xsl:if	Used to define the value of the <code>test</code> attribute. For example: <code><xsl:if test="authors/author= 'Henry'"/></code> See xsl:if and xsl:choose on page 475 to learn more about when to use this instruction.
xsl:choose	Used to define the value of the <code>test</code> attribute of the nested <code>xsl:when</code> element. For example: <code><xsl:when test="contains(authors/author, 'Marchese')"/></code> See xsl:if and xsl:choose on page 475 to learn more about when to use this instruction.

Table 57. XSLT Instruction Blocks in XSLT Mapper

<i>XSLT Instruction</i>	<i>Meaning of Input Port</i>
xsl:apply-templates	Used to define the value of the select attribute. For example: <xsl:apply-templates select="subject"/>
xsl:call-template	Used to define the value of the name attribute. For example: <xsl:call-template name="newAuthorsTemplate"/>

Specifying Values for Input Ports

You can specify values for input ports by:

- Dragging a link from a source document node or from the output port of another block (like that of an XPath function or If block, for example).
- Double-clicking the port and typing a value (a string or an XPath expression, for example) in the **Value** dialog box.

Tip When you mouse over an input port, Stylus Studio displays the value associated with that port.

Red Input Ports

If an xsl: instruction's attribute takes a literal or string value (such as xsl:value-of select="'Recommended'"/>, for example) and a value has been provided for the attribute, Stylus Studio fills the input port associated with that attribute with a deep red to indicate that a value has been specified.

The Flow Port

Output ports for any of the following xsl: instructions can be linked to the flow port of an instruction block:

- xsl:if
- The xsl:when element of xsl:choose
- xsl:for-each

You might decide you want a particular xsl:for-each instruction executed only after performing a certain function, for example.

Adding an Instruction Block to the XSLT Mapper

Tip If you enter an XSLT instruction in the XSLT source and that instruction can be graphically represented in the mapper, the instruction, including appropriate links to source and target nodes, appears the next time you display the **Mapper** tab.

◆ **To add an instruction block to the XSLT mapper**

1. Right click on the mapper canvas.
The shortcut menu appears.
2. Select **XSLT Instructions** from the shortcut menu.
The **XSLT Instructions** submenu appears.
3. Select the instruction you want to add to your XSLT.
The block for the instruction you selected appears in the mapper canvas.
4. Provide a value for the input port(s). See [Specifying Values for Ports](#) on page 472 if you need help with this step.
5. Link the output port(s).
6. Optionally, link the flow port.

Notes About Creating Instruction Blocks

Be aware of the following when working with XSLT instruction blocks in the XSLT mapper:

- To simplify the mapper's appearance, Stylus Studio sometimes removes blocks from the mapper canvas and replaces them with a simple link. For example, imagine creating an `xsl:value-of` instruction block and linking source and target document nodes. Stylus Studio displays the instruction block, but if you leave the **Mapper** and later return to it, the block symbol for the `xsl:value-of` instruction is replaced by a link with a small circle at its center, link the one shown in [Figure 229](#).



Figure 229. `xsl:value-of` in XSLT Mapper

If you mouse over the circle, Stylus Studio displays the XSLT represented by the link (`xsl:value-of select="/books/book/@pubdate"/`, for example).

This behavior also exists for text blocks created in the mapper that are not also linked to other blocks in the mapper. See [Setting a Text Value](#) on page 480.

- If you type an XSLT instruction in the XSLT source that is not represented by the XSLT mapper, no representation of that XSLT instruction is displayed on the **Mapper** tab. The code remains as part of the XSLT source, however.
- If you start creating an XSLT instruction in the mapper but do not completely define it (say you specify only the input port for an `xs1:for-each` instruction, for example), it is not represented in the XSLT source, and it is removed from the XSLT mapper if you leave the **Mapper** tab and then return to it.

xs1:if and xs1:choose

The `xs1:if` instruction cannot express an else condition. It has a single input port, and a single output port, as shown in [Figure 230](#).

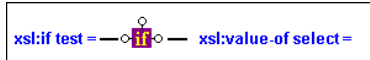


Figure 230. xs1:if Instruction Block

Once fully defined, the `xs1:if` block generates code like the following:

```
<Review>
  <xs1:if test="authors/author= 'Minollo'">
    <xs1:value-of select="'Recommended'"/>
  </xs1:if>
  <xs1:if test="contains(authors/author, 'Pedruzzi')">
    <xs1:value-of select="'A best buy'"/>
  </xs1:if>
</Review>
```

If you need to express an else condition, use the `xs1:choose` instruction block. This instruction block has two output ports by default, one for the `xs1:when test=` attribute, and one for the one `xs1:otherwise` element.

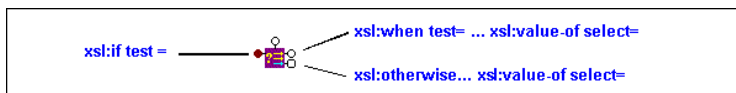


Figure 231. xs1:choose Instruction Block

The `xs1:choose` instruction block generates code like the following:

```
<Review>
  <xs1:choose>
    <xs1:when test="authors/author= 'Minollo'">
      <xs1:value-of select="'Recommended'"/>
    </xs1:when>
    <xs1:when test="contains(authors/author, 'Pedruzzi')">
      <xs1:value-of select="'authors best buy'"/>
    </xs1:when>
    <xs1:otherwise>
      <xs1:value-of select="'bah...'" />
    </xs1:otherwise>
  </xs1:choose>
</Review>
```

If you need to define more than one `xs1:when` `test=` attribute, use the **xs1:choose** shortcut menu (right click) and select **Add When Port**.

Note Stylus Studio generates the `xs1:otherwise` element by default for all `xs1:choose` instructions.

Processing Source Nodes

You can use any of the following to combine nodes or process nodes in the source document and map the result to a node in the target document:

- [XPath Function Blocks](#) on page 476
- [Logical Operators](#) on page 479
- [Setting a Text Value](#) on page 480
- [Defining Java Functions in the XSLT Mapper](#) on page 482

XPath Function Blocks

Stylus Studio supports standard XPath functions defined by the W3C. This section describes how to work with function blocks in XSLT mapper and covers the following topics:

- [“Parts of a Function Block”](#) on page 477
- [“Types of Function Blocks”](#) on page 477
- [“Creating a Function Block”](#) on page 479
- [“Deleting a Function Block”](#) on page 479

Parts of a Function Block

Function blocks are drawn as a purple block with an italic “*f*” at its center, and connectors, called *ports*, placed along the block’s border. Input ports (one or more depending on the function) on the left, the flow port at the top, and the output port on the right:

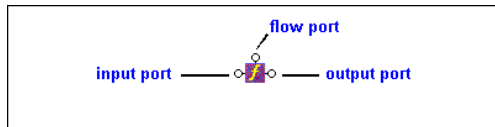


Figure 232. Function Block

Input ports

Input ports are on the left side of the function block. The number and definition of input ports varies from function to function. To specify a value for an input port, you can

- Drag a source document element or attribute to the port and release it
- Double-click the port and enter the function in the **Value** dialog box

Flow port

Flow ports on the top of function blocks are generally used only when a function is used in a direct link between a source and target node.

Output port

The output port is on the right side of the function block. You use the output port to map the function result directly to a target structure element or attribute, or to an IF, condition, or another function block.

Types of Function Blocks

The XPath functions available in XSLT mapper include the following:

- boolean
- ceiling
- concat
- contains
- count
- floor
- format

- last
- local-name
- mod
- name
- normalize-space
- number
- position
- round
- starts-with
- string
- string-length
- substring
- substring-after
- substring-before
- sum
- translate

XPath Mathematical Functions

In order to simplify the graphical presentation in the XSLT mapper, the following XPath mathematical functions are not graphically represented:

- add
- div
- multiply
- subtract

You can easily express these functions by typing them in the **Value** dialog box displayed when you double-click an input port.

Creating a Function Block

◆ To create a function block:

1. In the XSLT editor, in the **Mapper** tab, right-click mapper canvas.
2. In the shortcut menu that appears, click **XPath Functions** and slide to the submenu.
3. Click the function you want to use. Stylus Studio displays a function block for the function you selected.

Deleting a Function Block

◆ To delete a function block:

Select it and press the Delete key.

If the function block is part of a link, deleting the function block also deletes the link.

Logical Operators

The Stylus Studio XSLT mapper allows you to graphically define the following types of conditions:

- Equal (=)
- Less than (<)
- Greater than (>)
- Less than or equal to (<=)
- Greater than or equal to (>=)
- and (&)
- or (||)

All condition blocks have two input ports and a single output port, as shown in this example of a greater than block.



Figure 233. Greater Than Block

You can map the return port to a target structure element or attribute, or to the input port on an XSLT instruction, XPath function, or another condition block.

Setting a Text Value

You can set text values for target structure elements and attributes. You might want to do this if you are composing a mapping whose target structure contains an element or attribute that requires a fixed value, instead of using a value gathered from an input XML document.

Example

Here is the XSLT code Stylus Studio generates for the `Title` element when a text value is specified for it:

```
<Book>
  <Title>Confederacy of Dunces</Title>
</Book>
```

Stylus Studio displays a red letter **T** for nodes for which you define a text value:



Figure 234. Symbols for Nodes With Text Values

There are two ways to set a text value:

- On the mapper canvas
- On the target node

How to Set a Text Value on the Mapper Canvas

◆ **To set a text value on the mapper canvas:**

1. Right-click on the mapper canvas.
The shortcut menu appears.
2. Select **Text Block** from the shortcut menu.
The text block appears on the mapper.



Figure 235. Text Block

Double-click the text block to display the **Value** dialog box.

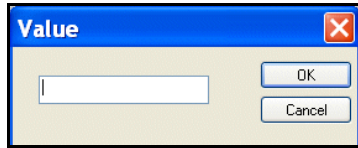


Figure 236. Value Dialog Box

Tip You can also display the **Value** dialog box by selecting **Properties** from the text block shortcut menu (right click).

3. Type a value and click **OK**.
4. Drag a link from the text block's output port to the target node to which you want to assign the text value.

Note Unless the text block is linked to another block (such as a logical operator or an XPath function), Stylus Studio removes it from the mapper canvas if you leave and then return to the **Mapper** tab. In this case, a red T is displayed next to the target node's name.

How to Set a Text Value on the Target Node

◆ To set a text value on a target node:

1. Right-click the node for which you want to set the text value.
The shortcut menu appears.
2. Select **Set Text Value** from the shortcut menu.
The **Value** dialog box appears.

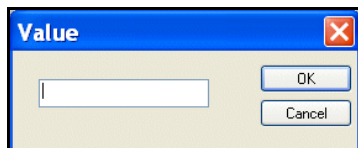


Figure 237. Value Dialog Box

3. Type the string you want to use as the text value and click **OK**.

Defining Java Functions in the XSLT Mapper

You can write your own Java functions and use them when you map nodes.

◆ **To define your own functions:**

1. Ensure that a Java virtual machine is running locally.
2. Create the class file for your function. See [“About Adding Java Class Files”](#) on page 482 for more help with this step.
3. Display the **Mapper** tab in the XSLT editor, if necessary.
4. Right-click the mapper canvas.
5. In the pop-up menu that appears, select **Java Functions > Register Java Extension Class**.
6. In the **Java Class Browser** dialog box that appears, navigate to and select the Java class that provides your function.
7. Click **OK** in the **Java Class Browser** dialog box.

Now when you select **Java Functions** from the mapper short-cut menu, the list of functions includes the function you registered.

About Adding Java Class Files

The class file must be in your CLASSPATH environment variable or in the Stylus Studio ClassPath. To add it to the Stylus Studio ClassPath, select **Tools > Options** from the Stylus Studio menu bar. In the **Options** dialog box, expand **Application Settings** and click **Java Virtual Machine**.

Creating and Working with Templates

A stylesheet can contain more than one template. This section describes Stylus Studio's features for creating and working with named and matched templates in XSLT mapper.

What Happens When You Create a Template

When you create a template, Stylus Studio switches the XSLT mapper to the new template. The attributes identifying the template you are currently viewing are displayed in the template drop-down list at the top of the mapper canvas.

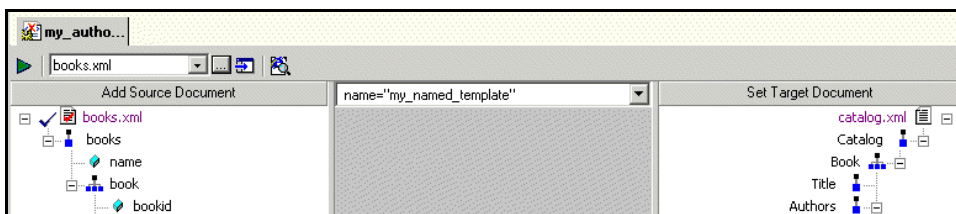


Figure 238. Drop-down List Shows Current Template

You can change the template view at any time, by selecting the template from the drop-down list, as shown in [Figure 239](#).

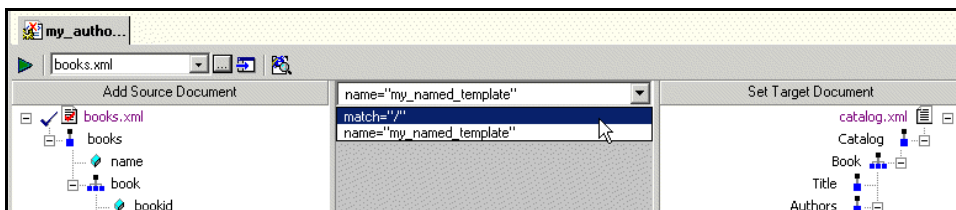


Figure 239. Display Different Templates Using the Drop-down List

Tip At any time, the mapper shows only the links that have been defined for the current template.

How to Create a Named or Matched Template

◆ **To create a named or matched template:**

1. Right-click the XSLT mapper canvas.
2. Select **Create Template > Named Template** or **> Matched Template** from the shortcut menu.

Stylus Studio displays the **Named Template** (or **Matched Template**) dialog box. (The **Named Template** dialog box is shown in [Figure 240](#).)

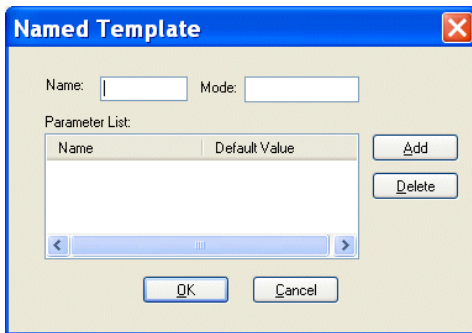


Figure 240. Named Template Dialog Box


3. Enter a name and, optionally, a mode.

Tip

You can use a mode to define the conditions under which a template will be applied by a stylesheet.

4. Optionally, create one or more parameters:
 - a. Click the **Add** button.
The **Name** column becomes editable.
 - b. Type a parameter name and press Enter.
The **Default Value** field becomes editable.
 - c. Type a default value.
 - d. If you want to define another parameter, click **ADD**; otherwise, go to [Step 5](#).
5. Click **OK** to finish creating the template.

Creating an XSLT Scenario

An *XSLT scenario* is a group of settings that Stylus Studio uses to process the XSLT when you click the **Preview Result** button (). Examples of scenario settings include the XML document to which the XSLT will be applied, whether you want to perform any post-processing, and the values of any parameters you might have defined. You can create multiple scenarios that use the same XSLT, and choose different settings for each. This flexibility can aid the XSLT development process as it enables you to easily test different applications of the XSLT before you put it online.

Even if you do not explicitly create a scenario, Stylus Studio uses default scenario settings in order to preview the XSLT. For example, Stylus Studio uses the first source document you specify as the document to which the XSLT is applied. (If the first source document is an XSD or DTD, Stylus Studio prompts you to provide an XML document for the scenario when you preview the XSLT.)

This section covers the following topics:

- [“Overview of Scenario Features”](#) on page 485
- [“How to Create a Scenario”](#) on page 489
- [“How to Run a Scenario”](#) on page 490
- [“How to Clone a Scenario”](#) on page 491

Overview of Scenario Features

This section describes the main features of XSLT scenarios. It covers the following topics:

- [“XML Source Documents”](#) on page 485
- [“Global Parameters”](#) on page 486
- [“XSLT Processors”](#) on page 488
- [“Performance Metrics Reporting”](#) on page 488
- [“Result Document Validation”](#) on page 489
- [“Post-Processing Result Documents”](#) on page 489

XML Source Documents

The main benefit of the XSLT scenario feature is that it lets you specify the XML document against which you want to run your XSLT. By default, Stylus Studio uses the first source document you add using the XSLT mapper as the XML source document for

the scenario. You can specify the XML source document setting on the **General** tab of the **Scenario Properties** dialog box.

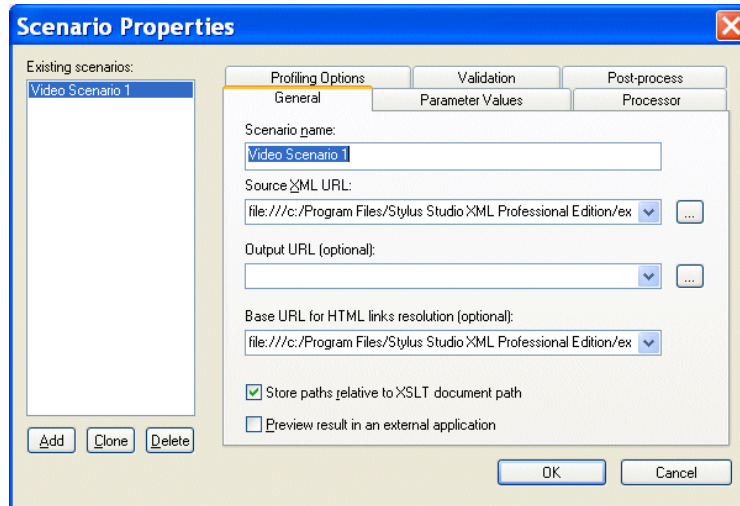


Figure 241. General Tab of XSLT Scenario Properties Dialog Box

See “[Source Documents](#)” on page 458 to learn more about the process of selecting and working with source documents in XSLT mapper.

Global Parameters

The **Parameter Values** tab of the **Scenario Properties** dialog box displays any global parameters you have defined in the XSLT source and allows you to

- Specify an alternate value to use when running the scenario
- Indicate whether the value is an XPath expression or a string

For example, imagine the following parameter defined in the XSLT source:

```
<xsl:param name="my_title" select="'Confederacy of Dunces'"/>
```

This parameter is displayed on the **Parameter Values** tab as follows:

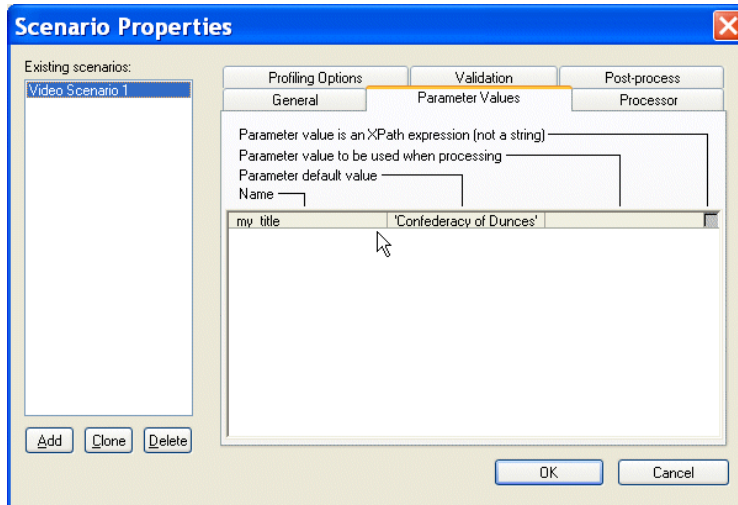


Figure 242. Specifying Alternate Values for XSLT Parameters

If you want to specify an alternate parameter value for this scenario, click the **Parameter value to be used when processing** entry field. If the alternate value you enter is an XPath expression, click the associated check box.

By default, values entered in the **Parameter value to be used when processing** field are interpreted as strings. However, you can indicate that you want the value to be interpreted as an XPath expression by selecting the check box in the **Parameter value is an XPath expression (not a string)** field. This allows you to enter expressions such as the following:

```
document("books.xml")/books/book[1]
```

Tip All global parameters you define for a stylesheet are displayed on the **Params/Other** tab of the XSLT Editor. Parameters displayed on the **Params/Other** tab are read-only.

XSLT Processors

By default, Stylus Studio uses a built-in processor to process XSLT documents. You can change the processor on the **Processor** tab of the **Scenario Properties** dialog box.

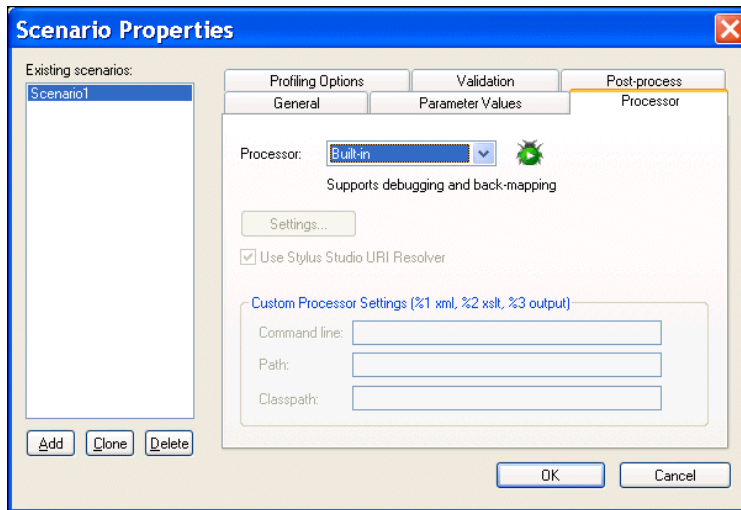


Figure 243. Changing the Default XSLT Processor

You can choose from a number of third-party processors that are bundled with Stylus Studio, or you can specify your own custom processor.

Note Not all third-party processors, including any custom processors you might specify, support back-mapping and debugging.

See [“How to Use a Third-Party Processor”](#) on page 388 for more information.

Performance Metrics Reporting



Performance metrics reporting is available only in Stylus Studio XML Enterprise Suite.

Stylus Studio can generate an HTML report that contains information about how your XSLT is being processed. This option is off by default, but you can enable it, and choose options for the report, on the **Profiling Options** tab.

See [“Profiling XSLT Stylesheets”](#) on page 500 to learn more about the different ways in which Stylus Studio can provide you with XSLT performance metrics.

Result Document Validation

You can optionally validate the XML document that results from XSLT processing using the XML validator you specify. You can use

- The Stylus Studio built-in XML validator. If you use the Stylus Studio built-in processor, you can optionally specify one or more XML Schemas against which you want the result document to be validated.
- Any of the customizable processors supported by Stylus Studio, such as the .NET XML Parser and XSV.

All validation is done before any post-processing that you might have specified.



See “[Validating Result Documents](#)” on page 392.

Post-Processing Result Documents

You can use the **Post-process** tab to specify any optional processing you want performed on the XML after it has been processed by the XSLT. You might want to use FOP to render XML as PDF, for example.

How to Create a Scenario

◆ To create a scenario:

1. In the XSLT Editor tool bar, click .
Alternative: Select **Create Scenario** from the scenario drop-down list at the top of the editor window.
Stylus Studio displays the **Scenario Properties** dialog box.
2. In the **Scenario name:** field, type the name of the new scenario.
3. In the **Source XML URL (optional):** field, type the name of the XML file to which you want to apply the XSLT, or click **Browse**  to navigate to an XML file and select it.

Note

If the first document you added to the XSLT mapper is an XML document, Stylus Studio uses that document as the XML source for the scenario and displays it in this field.



4. In the **Output URL** field, optionally type or select the name of the result document you want the XSLT document to generate. If you specify the name of a file that does not exist, Stylus Studio creates it when you preview the XSLT.

5. If you want Stylus Studio to **Store paths relative to XSLT document path**, ensure that this option is checked.
6. If you check **Preview result in an external application**, Stylus Studio displays the result in Internet Explorer. In addition, Stylus Studio always displays XSLT results in the **Preview** window.
7. Optionally, configure settings for global parameters, the XSLT processor you want to use, whether or not you want to run a profiling report, and whether or not you want to perform any post-processing on the XSLT result. See [Overview of Scenario Features](#) on page 485 for more information.
8. To define another scenario, click **Add** and enter the information for that scenario. You can also copy scenarios. See [How to Clone a Scenario](#) on page 491.
9. Click **OK**.

If you start to create a scenario and then change your mind, click **Delete** and then **OK**.

How to Run a Scenario


◆ To run a scenario:

1. Select a scenario from the scenario drop-down list at the top of the editor window.
Alternative:
 - a. In the XSLT Editor tool bar, click .
Stylus Studio displays the **Scenario Properties** dialog box.
 - b. On the **General** tab, select the scenario you want to run from the **Existing Scenarios** list.
 - c. Click **OK**.
2. Click the **Preview Result** button ().

How to Clone a Scenario

When you clone a scenario, Stylus Studio creates a copy of the scenario except for the scenario name. This allows you to make changes to one scenario and then run both to compare the results.

◆ **To clone a scenario:**

1. Display the XSLT in the scenario you want to clone.
2. In the XSLT editor tool bar, click  to display the **Scenario Properties** dialog box.
3. In the **Scenario Properties** dialog box, in the **Existing preview scenarios** field, click the name of the scenario you want to clone.
4. Click **Clone**.
5. In the **Scenario name** field, type the name of the new scenario.
6. Change any other scenario properties you want to change. See [How to Create a Scenario](#) on page 489.
7. Click **OK**.

If you change your mind and do not want to create the clone, click **Delete** and then **OK**.

Chapter 6 **Debugging Stylesheets**

Stylus Studio provides several tools that allow you to follow XSLT processing and detect errors in your stylesheets. To use these tools, you must use the processors displayed in the **Debug and back-mapping enabled** section of the **Processors** page on the **Scenario Properties** dialog box. These processors include Stylus Studio's XSLT processor, MSXML .Net, and Saxon 6 and 8. If you use the MSXML XSLT processor or some other XSLT processor, you cannot use the Stylus Studio debugging and backmapping tools.

You can also use the Stylus Studio debugger to analyze Java files, or applications that include both stylesheets and Java files. In addition, you can use Stylus Studio to debug JavaScript and VBScript extension functions. See the Microsoft documentation for information about these extension functions.

This section discusses the following topics:

- [“Steps for Debugging Stylesheets”](#) on page 494
- [“Using Breakpoints”](#) on page 494
- [“Viewing Processing Information”](#) on page 495
- [“Using Bookmarks”](#) on page 498
- [“Determining Which Template Generated Particular Output”](#) on page 499
- [“Determining the Output Generated by a Particular Template”](#) on page 499
- [“Profiling XSLT Stylesheets”](#) on page 500
- [“Handling Parser and Processor Errors”](#) on page 503
- [“Debugging Java Files”](#) on page 503

Steps for Debugging Stylesheets

Stylus Studio provides tools for debugging transformations.

◆ **To debug a stylesheet:**

1. Open a stylesheet.
2. [Set up a scenario](#) or select the scenario you want to use. See “[Applying Stylesheets](#)” on page 366.
3. [Set one or more breakpoints](#). See “[Using Breakpoints](#)” on page 494.
4. [Apply the stylesheet](#) by pressing F5, not clicking **Preview Result**. If you click **Preview Result**, Stylus Studio applies the stylesheet without invoking the debugger.
5. [Examine the information in the debugging tools](#) and in the **Preview** window.
6. [Run and examine the information XSLT Profiler report](#).
7. Iteratively [step through](#) the stylesheet or program and examine the information in the debugging tools.

You can include `msxml:script` elements in XML documents in Stylus Studio. The `msxml` prefix must indicate the Microsoft `urn:schemas-microsoft-com:xslt` namespace.



The following sections provide the details for performing each of these steps.

Using Breakpoints

The Stylus Studio debugger allows you to interrupt XSLT or Java processing to gather information about variables and processor execution at particular points.

Inserting Breakpoints


◆ **To insert a breakpoint:**

1. In the XSLT stylesheet or Java file in which you want to set a breakpoint, place your cursor where you want the breakpoint to be.
2. Click **Toggle Breakpoint**  or press F9. Stylus Studio inserts a blank stop sign  to the left of the line with the breakpoint.


Removing Breakpoints






◆ To remove a breakpoint:


1. Click in the line that has the breakpoint.
2. Press F9 or click **Toggle Breakpoint**.

Alternative: In the Stylus Studio tool bar, click  **Breakpoints** to display a list of breakpoints in all open files. You can selectively remove one or more, remove them all, or jump to one of them.

Start Debugging

When your stylesheet or Java file has one or more breakpoints set, start processing by clicking **Start Debugging**  or pressing F5. When Stylus Studio reaches the first breakpoint, it suspends processing and activates the debugging tools. After you examine the information associated with that breakpoint (see “[Viewing Processing Information](#)” on page 495) you can choose to

- Step into. Click  or press F11.
- Step over. Click  or press F10.
- Step out. Click  or press Shift+F11.
- Run to cursor. Click .
- Continue processing. Press F5.
- Stop processing. Click **Stop Debugging**  in the Stylus Studio tool bar, or click **Cancel** in the lower right corner of the stylesheet editor, or press Shift+F5.

Note You can also click **Pause**  to suspend XSLT processing. Stylus Studio flags the line it was processing when you clicked **Pause**.


Viewing Processing Information

Stylus Studio provides several tools for viewing processing information when you suspend processing. The tools become active when processing reaches a breakpoint. This section discusses the following topics:

- [Watching Particular Variables](#) on page 496
- [Evaluating XPath Expressions in the Current Processor Context](#) on page 496
- [Obtaining Information About Local Variables](#) on page 496

- [Determining the Current Context in the Source Document](#) on page 497
- [Displaying a List of Process Suspension Points](#) on page 497
- [Displaying XSLT Instructions for Particular Output](#) on page 498

Watching Particular Variables


Use the **Watch** window to monitor particular variables. To display the **Watch** window, click **Watch**  in the Stylus Studio tool bar. This button is active when Stylus Studio suspends processing because it reached a breakpoint. Stylus Studio displays the **Watch** window only when processing is suspended.

Enter the names of the variables you want to watch. You can enter as many as you like. In a Java program, you can double-click a symbol and drag it to the **Watch** window to enter it as a variable you want to watch. When Stylus Studio suspends processing, it displays the current values for any variables listed in the **Watch** window. You can expand and collapse complex structures as needed.


Another way to obtain the value for a variable is to hover over the symbol in your stylesheet or Java program. Stylus Studio displays a pop-up box that contains the current value.

During XSLT debugging, you can enter XPath expressions in the **Watch** window fields. Stylus Studio uses the current context to evaluate these expressions, and displays the results with the same kind of interface Stylus Studio uses for `nodeList` and `node` variables.

Evaluating XPath Expressions in the Current Processor Context

When you suspend processing, you can evaluate an XPath expression in the context of the suspended process. You do this in the **Watch** window. Click  in the Stylus Studio tool bar to display the **Watch** window. Click in an empty name field and enter an XPath expression. As soon as you press Enter, Stylus Studio displays the results of the evaluation in the **Value** field of the **Watch** window.

Obtaining Information About Local Variables

Display the **Variables** window to obtain information about local variables. To display the **Variables** window, click **Variables**  in the Stylus Studio tool bar. This button is active when Stylus Studio suspends processing because it reached a breakpoint. Stylus Studio displays the **Variables** window only when processing is suspended.

For stylesheets, Stylus Studio displays

- A path that shows which node in the stylesheet was being processed when processing was suspended
- Local and global XSLT parameter values
- Local and global XSLT variable values

Also, you can navigate the structure associated with a variable, a parameter, or the current context if it is a node list or a node.

For Java classes, Stylus Studio displays


- Local variables that are defined at that point in the processing and their values.
- Function parameters and their values.
- A special variable named `this`. The `this` variable represents the object being processed. It allows you to drill down and obtain additional information.

You can expand and collapse complex structures as needed.

Determining the Current Context in the Source Document

When you are debugging a stylesheet, the **Variables** window displays a path for the current context. This is the set of nodes that the XSLT processor is currently working through. This allows you to examine the nodes that lead to the context node.

Displaying a List of Process Suspension Points

Display the **Call Stack** window to view a list of the locations at which processing was suspended. To display the **Call Stack** window, click **Call Stack**  in the Stylus Studio tool bar. This button is active when Stylus Studio suspends processing because it reached a breakpoint. Stylus Studio displays the **Call Stack** window only when processing is suspended.

For stylesheets, Stylus Studio displays the template name and line number. For Java classes, Stylus Studio displays the class name, function name, parameters, and line number.

When processing is complete, the call stack is empty.

When execution is suspended you can use the **Call Stack** window to jump directly to the XSLT or Java source. Double-click on a stack line to go to that location. A green triangle appears to indicate this location in the source file.

The **Call Stack** window and the **Backmap Stack** window provide the same kind of information. However, the **Backmap Stack** window never shows Java entries, and the contents of the **Backmap Stack** window can be different from the **Call Stack** window according to where you click in the output to enable backmapping.

Displaying XSLT Instructions for Particular Output

After you apply a stylesheet, or during debugging of a stylesheet, Stylus Studio can display the XSLT instruction or the sequence of XSLT instructions that generate a particular part of a result document. This can be particularly helpful when the result is not quite what you want.

◆ To view XSLT instructions:

1. Open a stylesheet.
2. Apply the stylesheet.
3. In the **Preview** window, in either the text view or the browser view, click on the output for which you want to display the XSLT calls.

Stylus Studio displays the **Backmap Stack** window, which lists one or more XSLT instructions. Also, Stylus Studio flags the line in the stylesheet that contains the first instruction in the list. To find the location of another listed instruction, click that instruction in the **Backmap Stack** window.


The **Call Stack** window and the **Backmap Stack** window provide the same kind of information. However, the **Backmap Stack** window never shows Java entries, and the contents of the **Backmap Stack** window can be different from the **Call Stack** window according to where you click in the output to enable backmapping.

Using Bookmarks

When you are editing or debugging a long file, you might want to repeatedly check certain lines in the file. To quickly focus on a particular line, insert a bookmark for that line. You can insert any number of bookmarks. You can insert bookmarks in any document that you can open in Stylus Studio.

◆ To insert a bookmark:

1. Click in the line that you want to have a bookmark.

2. Click **Toggle Bookmark**  in the Stylus Studio tool bar. Stylus Studio inserts a turquoise box with rounded corners to the left of the line that has the bookmark.

◆ **To remove a bookmark:**

1. Click in the line that has the bookmark you want to remove.
2. Click **Toggle Bookmark** in the Stylus Studio tool bar. Stylus Studio removes the turquoise box.

◆ **To remove all bookmarks in a file, click Clear All Bookmarks** .

◆ **To move from bookmark to bookmark, click Next Bookmark**  **or Previous Bookmark** .

Determining Which Template Generated Particular Output

In Stylus Studio, you can easily determine which template is responsible for generating any particular portion of the HTML output.

Click anywhere in the **Preview in tree**, **Preview in Browser**, or **Preview Text** view of the **Preview** window. In the **XSLT Source** tab, Stylus Studio points to the line that generated that portion of the HTML output. This is the Stylus Studio backmapping feature.

Note It is possible for backmapping to point to the wrong line if you made changes in the XSLT source and did not preview the results.

Determining the Output Generated by a Particular Template

In the **XSLT Source** pane, if the cursor is in a template, the output from that template has a gray background in the **Preview Text** view of the **Preview** window. In the **Preview in tree** view of the **Preview** window, the contents generated by the template are highlighted.

In the **Preview in Browser** view of the **Preview** window, there is no gray shading to indicate the output from the currently displayed template.

Profiling XSLT Stylesheets



The XSLT Profiler is available only in Stylus Studio XML Enterprise Suite.

In addition to debugging tools for XSLT, Stylus Studio provides the *XSLT Profiler*, a tool that helps you evaluate the efficiency of your XSLT. By default, the performance metrics gathered by the XSLT Profiler are displayed in a preformatted report, like the one shown [Figure 244](#):

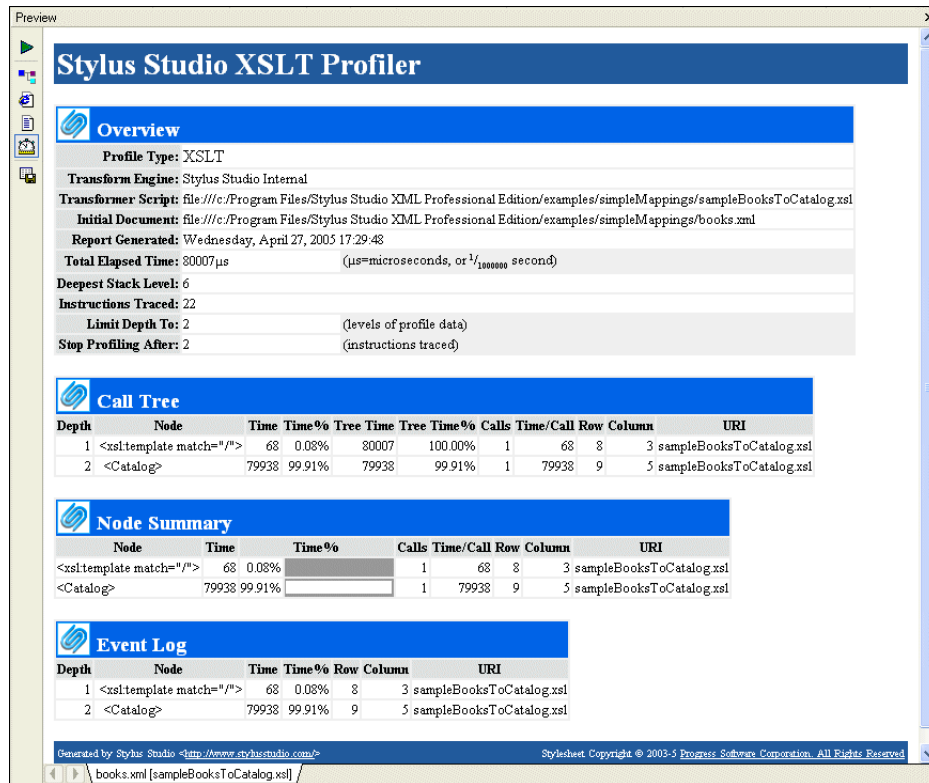


Figure 244. XSLT Profiler Report

The report format is controlled by the default XSLT stylesheet, **profile.xsl**, in the \Stylus Studio\bin directory. You can customize this stylesheet as required. You can save XSLT Profiler reports as HTML.

Note XSLT and XQuery Profiler reports use the same XSLT stylesheet.

In addition to generating the standard XSLT Profiler report, you can save the raw data generated by the Profiler and use this data to create your own reports. See “[Enabling the Profiler](#)” on page 501 for more information about this procedure.



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the XSLT Profiling video](#).

A complete list of all the videos demonstrating Stylus Studio’s features is here: http://www.stylusstudio.com/xml_videos.html.

About Metrics

The XSLT Profiler can record three different levels of performance metrics:


- A call tree of execution times
- Execution times by XSLT element, and
- A detailed log of step-by-step element execution

Note Displaying the report for a step-by-step log can take significantly longer than evaluating the XSLT itself. Consider using the Profiler with the first two performance metric options. You can also use the **Limit Trace To** fields to further restrict the Profiler’s scope. If you find you need still more detail (while troubleshooting a performance bottleneck, for example), use the step-by-step setting.

Enabling the Profiler

The XQuery Profiler is off by default. You enable the Profiler on the **Profiling Options** tab of the XSLT **Scenario Properties** dialog box.

◆ To enable the XSLT Profiler:

1. Open the **Scenario Properties** dialog box for the XSLT stylesheet. (Click **Browse**  at the top of the XSLT editor window.)

2. Click the **Profiling Options** tab.

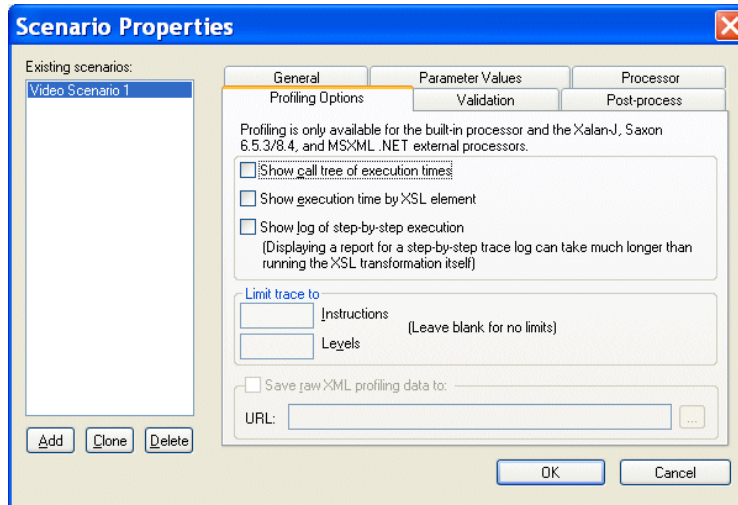


Figure 245. Profiling Options

3. Select the settings for the performance metrics you want the Profiler to capture.
4. Optionally, save the raw Profiler data to a separate file.

Note

This option is available only after you select one or more performance metrics settings.

5. Click OK.

The next time you preview the XSLT results, the performance metrics you selected are available to you in the XSLT Profiler report (and as raw data if you selected that setting and specified a file).

Displaying the XSLT Profiler Report

◆ **To display the XSLT Profiler report:**

1. Ensure that the Profiler is enabled. (See “[Enabling the Profiler](#)” on page 501 if you need help with this step.)
2. Click the **Preview Result** button (▶).
3. Click the **Show Profiling Report** button (🕒).

The XSLT Profiler report appears in the **Preview** window.

Handling Parser and Processor Errors


When you refresh stylesheet output, Stylus Studio parses and processes your XML document and XSLT stylesheet. If the processor encounters a parser or processing error, Stylus Studio displays a message that indicates the nature and location of the error. Stylus Studio prompts you to indicate if you want to jump to the error location in your stylesheet.

Debugging Java Files



Support for debugging Java extensions is available only in Stylus Studio XML Enterprise Suite and Stylus Studio XML Professional Suite.

The Stylus Studio debugger allows you to follow Java processing as well as XSLT processing. With the Stylus Studio debugger, you can observe the interaction between your Java code and XML data.

When you debug a transformation, the transformation can include the processing of Java files. Such Java files might be servlets, server extensions, extension functions, or other kinds of Java programs that involve stylesheets. If you need to make a change to your Java file, you can compile it right in Stylus Studio. Click **Compile**  in the upper left corner of the Java file window. Stylus Studio automatically saves the file before it compiles it.

This section discusses the following topics:

- [Requirements for Java Debugging](#) on page 503
- [Setting Options for Debugging Java](#) on page 504
- [Using the Java Editor](#) on page 505
- [Stylus Studio and the JVM](#) on page 506
- [Example of Debugging Java Files](#) on page 506

Requirements for Java Debugging

If you want to use Stylus Studio to debug Java code, you must have the Sun Java Runtime Environment (JRE) 1.4.x installed. If you want to use Stylus Studio to assist you in editing and compiling Java code, you must have the Sun JDK 1.4.x installed.

You can download the Sun Java products from <http://www.javasoft.com/j2se/>.

After you install the JRE, you must run the Stylus Studio auto-detect feature. For more information, see “[Configuring Java Components](#)” on page 134.

◆ **To run the auto-detect feature:**

1. Select **Tools > Options** from the Stylus Studio menu bar.
2. In the **Options** dialog box, click **Java Virtual Machine**.
3. In the **Java Virtual Machine** page, click **Auto detect**.

Also, in the **Parameters** field of the Java Virtual Machine page, there should be something like the following:

```
-Xdebug -Xnoagent -Xrunjdwp:transport=dt_socket,  
server=y,suspend=n,address=8000 -Djava.compiler=NONE
```

To confirm that your set up is correct, select **Help > About Stylus Studio** from the Stylus Studio menu bar. The **Java Virtual Machine** field in the **About Stylus Studio** dialog box should indicate that the JVM is running in debug mode.

Setting Options for Debugging Java

You can specify the following options when you use Stylus Studio to debug Java code:

- **Source Path** is the path that Stylus Studio uses to locate the source files during debugging.
- **Prompt user for source file path confirmation** indicates that if Stylus Studio cannot find the source files you are debugging, it prompts you to specify the source file path. If you do not set this option, and Stylus Studio cannot find a source file, the behavior varies according to what the debugger is trying to do. For example, if the debugger is stepping into an instruction that calls a function that is defined in a Java file that Stylus Studio cannot find, the debugger steps over the instruction.
- **Never step into classes starting with one of the following strings of characters:** contains a list of classes, one on each line, that you do not want to step into. For example, these might be classes that are part of the core language, or classes that you do not have source files for. If you specify `java.lang`, the debugger skips all classes whose names start with `java.lang`, for example, `java.lang.String` and `java.lang.Object`.
- **JVM communication time out** indicates the amount of time that Stylus Studio waits for a response from the JVM. If Stylus Studio does not receive a response in the specified amount of time, it stops trying to communicate with the JVM. The default is 5 seconds.

- **Show JDWP Events in the Output Window** indicates whether you want Stylus Studio to log all communication events in the Stylus Studio **Output Window** during a debugging session.

Stylus Studio also allows you to set options that specify the Java virtual machine (JVM) you use. You can specify the run-time library, the home directory, and parameters for starting the JVM. Select **Tools > Options** from the Stylus Studio menu bar.

See “[Specifying Stylus Studio Options](#)” on page 121 for instructions for setting these options.

Using the Java Editor

To use the Stylus Studio Java editor, open a Java file in Stylus Studio.

To specify arguments that Stylus Studio uses to run the active Java class, select **Java > Class Properties** from the Stylus Studio menu bar. Stylus Studio displays the **Class Properties** dialog box. Enter the arguments required to run your code. (You must have a Java file open in Stylus Studio for **Java** to appear in the menu bar.)

The same debugging capabilities that are available when you are debugging XSLT stylesheets are available when you are debugging stand-alone Java applications.

When you use the Java editor, the Sense:X auto-completion feature is available. The Java editor browses your import directives to gather information about the packages you are using and provides auto-completion when using methods or data members defined in imported classes. The auto-completion mechanism also provides you with tips about the signature of the class method and its required arguments. The same applies to the classes that you are editing. Also, the CLASSPATH is used to help you auto-complete import directives. Type Ctrl+Space if you want Stylus Studio to auto-complete keywords and class names that are defined in java.lang.package.


The Stylus Studio Java editor also does background error checking. As you type Java code, Stylus Studio displays red lines that indicate syntax errors. Move the cursor over the red line to display a pop-up error message.

When you use the Java editor, you can configure the character encoding that Stylus Studio uses to save and load files. To do this, ensure that a Java file is the active file. Then select **Edit > Change Encoding** from the Stylus Studio menu bar.

Context-sensitive help for your Java classes is available in the Java editor. The directory that contains the javadoc-generated documentation must be in the Stylus Studio class path (in the Stylus Studio menu bar select **Tools > Options > Java Virtual Machine**) or in your

CLASSPATH environment variable. You can then press F1 when your cursor is on a class name in the Java editor. Stylus Studio opens the related javadoc-generated documentation.

Stylus Studio and the JVM


Stylus Studio allows you to debug a running application. You can attach Stylus Studio to a local or remote JVM, and run your application in debug mode. In the Stylus Studio tool bar, click **Attach**  to debug a standalone Java program that is running an external JVM. (**Attach** is not for debugging Java extensions.)

To execute a class, open the Java source in Stylus Studio and press F5. Of course, the class must be in your CLASSPATH environment variable or in the Stylus Studio ClassPath (select **Tools > Options > Java Virtual Machine**).

◆ **To verify the JVM that Stylus Studio is trying to load:**

1. Select **Tools > Options** from the Stylus Studio menu bar.
2. In the **Options** dialog box that appears, click **General > Java Virtual Machine**.

The **Home Directory** field indicates the version of the JVM.

When you suspend processing, display the **Output Window** to view any output from the Java virtual machine. To display the **Output Window**, click **Output Window**  in the Stylus Studio tool bar.

Example of Debugging Java Files

Stylus Studio includes sample files that you can experiment with to learn how to use the debugger with an application that includes stylesheets and Java files. To get you started, this section provides step-by-step instructions for using the debugger with these sample files. You should perform the steps in each topic in the order of the topics.



For complete information about how to use the debugger, see “[Debugging Stylesheets](#)” on page 493.

This section includes the following topics:

- [Setting Up to Debug Sample Java/XSLT Application](#) on page 507
- [Inserting a Breakpoint in the Sample Java/XSLT Application](#) on page 507
- [Gathering Debug Information About the Sample Java/XSLT Application](#) on page 508



Setting Up to Debug Sample Java/XSLT Application

◆ To set up Stylus Studio to debug the sample Java/XSLT application:

1. From the Stylus Studio menu bar, select **Tools > Options**.
2. In the **Options** dialog box that appears, click **Java Virtual Machine**.
3. If the `examples\javaExtension` directory is already in the **ClassPath** field, click **OK**.
If the `examples\javaExtension` directory is not in the **ClassPath** field, click **Browse**  next to the **ClassPath** field. In the **Browse for Folder** dialog box that appears, navigate to and select the `javaExtension` directory, which is in the `examples` directory of your Stylus Studio installation directory. Click **OK**. Restart Stylus Studio. For **ClassPath** changes to take effect, you must restart Stylus Studio whenever you modify the **ClassPath** field.
4. In the File Explorer, navigate to the `examples\javaExtension` directory in your Stylus Studio installation directory.
5. Double-click `IntDate.xsl`.
Stylus Studio opens the `IntDate.xsl` stylesheet in the XSLT editor. The tree for the XML source document, `IntDate.xml`, also appears.
6. In the XSLT editor tool bar, click **Preview Result**  .
The `IntDate` scenario has already been defined. Stylus Studio applies the stylesheet and displays the results (a list of dates) in the **Preview** window.

Inserting a Breakpoint in the Sample Java/XSLT Application





To insert a breakpoint in the sample stylesheet:

1. In the XSLT editor, examine the template that matches the date element.
As you can see, the `select` attribute in the `xsl:value-of` instruction invokes the `IntDate` Java extension function.
2. In the body of the template, click just before the **xsl:value-of** instruction.
3. In the Stylus Studio tool bar, click **Toggle Breakpoint**  .
4. Press F5 to apply the stylesheet.
Alternative: In the Stylus Studio tool bar, click **Start Debugging**  .

The XSLT processor suspends processing at the breakpoint, displays a yellow triangle to indicate where processing has been suspended, and displays a message in the **Preview** window.

Gathering Debug Information About the Sample Java/XSLT Application

◆ **To obtain debug information:**

1. In the Stylus Studio tool bar, click **Step into**  or press F11.
Stylus Studio opens and displays the Java source file that contains the `IntDate` extension function. Now the **Variables** window displays a list of the variables in the extension function. There is still no output in the **Preview** window.
Stylus Studio might display the **Browse For Folder** dialog box. It is prompting you to specify where it can find the Java source file that contains the extension function invoked in the line that has the breakpoint. Stylus Studio does not display the **Browse for Folder** dialog box when the `.java` file is in the same directory as the `.class` file. Click the `javaExtension` directory and click **OK**.
2. In the Stylus Studio tool bar, click **Output Window** .
Stylus Studio displays the **Output Window**, which displays output from the Java virtual machine.
3. In the Stylus Studio tool bar, click **Step Over**  or F10 to move to the next line of Java code.
The yellow triangle moves to show the new location. If the values of the variables change, the **Variables** window reflects this.
4. Press **Step Out**  to return to the stylesheet.
The **Variables** window now displays only the context node. Processing was suspended when the second date child element of the `doc` document element was the context node.
The **Preview** window now displays a few lines of HTML.
5. In the **Preview** window, click in the first line of text.
Stylus Studio displays the **Backmap Stack** window, which contains a list of the XSLT instructions that have been executed. Also, in the **XSLT Source** tab, Stylus Studio displays a blue triangle that indicates the line in the stylesheet that generated the output line you clicked in.

Chapter 7 **Defining XML Schemas**

This section provides information about how to use Stylus Studio to define an XML Schema. Although some information about XML Schema tags is provided, familiarity with the W3C *XML Schema Recommendation* is assumed.

Many of the examples in this chapter are based on the `purchaseOrder.xsd` document, which is installed with other sample files in the `examples\simpleMappings` directory of your Stylus Studio installation directory. Consider having this file open as you read through the examples in this chapter.

This section covers the following topics:

- “What Is an XML Schema?” on page 510
- “Creating an XML Schema in Stylus Studio” on page 510
- “Working with XML Schema in Stylus Studio” on page 521
- “Getting Started with XML Schema in the Tree View” on page 530
- “Defining simpleTypes in XML Schemas” on page 536
- “Defining complexTypes in XML Schemas” on page 546
- “Defining Elements and Attributes in XML Schemas” on page 556
- “Defining Groups of Elements and Attributes in XML Schemas” on page 565
- “Adding Comments, Annotation, and Documentation Nodes to XML Schemas” on page 569
- “Defining Notations” on page 572
- “Referencing External XML Schemas” on page 573
- “Generating Documentation for XML Schema” on page 580
- “Generating JAXB Classes” on page 585
- “About XML Schema Properties” on page 587

What Is an XML Schema?

An XML Schema conforms with the W3C *XML Schema Recommendation*. The *XML Schema Recommendation* defines an XML markup vocabulary for specifying the structure of an XML document. An XML Schema serves the same purpose as a DTD. The most visible difference is that an XML Schema is in XML, while a DTD is not.

Like a DTD, an XML Schema describes the structure of a document. However, an XML Schema contains more specialized types of nodes than a DTD schema. For example, in an XML Schema, you can define nodes of type `group` and `attributeGroup`. These nodes contain groups of elements and attributes, respectively.

In an XML Schema, elements that contain subelements or attributes are called *complexType*s. Elements that contain data but do not contain subelements or attributes are *simpleTypes*. Attributes are always *simpleTypes*. In your XML Schema, along with elements and attributes, you define *complexType*s and some *simpleTypes*. In addition, many *simpleTypes* are part of the XML Schema grammar.

Reference Information

The World Wide Web Consortium (W3C) provides information about XML Schema, including the following:

- *XML Schema Part 0: Primer* at <http://www.w3.org/TR/xmlschema-0/>
- Glossary of XML Schema terms at <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/#normative-glossary>
- Reference information for *simpleTypes* and their facets at <http://www.w3.org/TR/xmlschema-0/#SimpleTypeFacets>
- Reference information for XML Schema elements and attributes at <http://www.w3.org/TR/xmlschema-0/#index>

Creating an XML Schema in Stylus Studio

There are several ways to create an XML Schema in Stylus Studio, including building your own XML Schema from scratch, and creating an XML Schema based on an existing DTD or from an XML document.

This section covers the following topics:

- “[Creating Your Own XML Schema](#)” on page 511
- “[Creating XML Schema from a DTD](#)” on page 511

- “Creating XML Schema from an XML Document” on page 516

You can also create XML Schema from EDI message types and transactions, like those in EDIFACT, X12, and IATA dialects. See “Creating XML Schema from EDI” on page 519.

Creating Your Own XML Schema

- ◆ **To create an XML Schema, select File > New > XML Schema from the menu bar.**

Stylus Studio displays a new XML document in the XML Schema Editor **Diagram** tab; the text pane displays the following contents:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
</xsd:schema>
```

When you create an XML Schema in Stylus Studio, the default namespace is specified as `http://www.w3.org/2001/XMLSchema`. If you choose to you specify the XML Schema namespace, be sure to specify one of the following:

- `http://www.w3.org/2001/XMLSchema`
- `http://www.w3.org/2001/XMLSchema-instance`

Creating XML Schema from a DTD

Stylus Studio has two document wizards you can use to create an XML Schema from a DTD. One uses a built-in processor; the other uses the Trang schema converter from Thai Open Source Software Center (www.thaiopensource.com). Using the Trang converter gives you more control over both the input file and output characteristics (such as whether or not you want to indent the XML Schema).

Using the DTD to XML Schema Document Wizard

- ◆ **To use the DTD to XML Schema wizard:**
 1. From the Stylus Studio menu bar, select **File > Document Wizards**. The **Document Wizards** dialog box appears.

- In the **XML Editor** tab, click **DTD to XML Schema**, and click **OK**.
The **Convert DTD to XML Schema** dialog box appears.

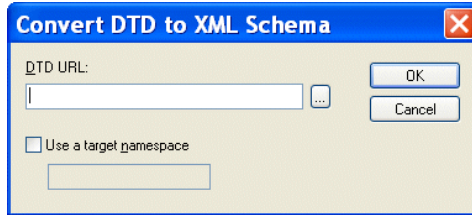


Figure 246. Convert DTD to XML Schema Dialog Box

- In the **DTD URL** field, type or select the absolute path for the DTD from which you want to create an XML Schema.

Note The DTD must be encoded in UTF-8.

- If you want, select the **Use a Target Namespace** check box and type a target namespace.
- Click **OK**.
Stylus Studio displays the new XML Schema in the XML Schema Editor.

Using the DTD to XML (Trang) Document Wizard

The following table describes the fields in the **DTD to XML (Trang)** dialog box, which is displayed when you run the DTD to XML (Trang) document wizard. This document wizard was created using Stylus Studio Custom Document Wizard (see [“Custom Document Wizards”](#) on page 1020 for more information).

Table 58. DTD to XML (Trang) Document Wizard Fields

<i>Field</i>	<i>Description</i>
Input file (required)	The name and location of the DTD file you want to convert to XSD. You can type the file name or use the Browse button to browse a file system for the source DTD file.
<code>[input] xmlns=<uri></code>	The default namespace; used for unqualified element names.

Table 58. DTD to XML (Trang) Document Wizard Fields

<i>Field</i>	<i>Description</i>
[input] xmlns:<prefix=uri>	The namespace for the element and attribute names using <i>prefix</i> .
[input] colon-replacement=<chars>	The character that is used to replace colons in element names. Used when constructing the names of definitions used to represent the element and attribute list declarations in the DTD. Trang generates a definition for each element declaration and attribute list declaration in the DTD. The definition name is based on the element name. In RELAX NG, the definition names cannot contain colons; colons are allowed in element names in DTDs. Trang first tries to use the element names without prefixes. If this results in a conflict, Trang replaces the colon with the chars specified. If no chars is specified, a period is used.
[input] element-define=<name-pattern>	Specifies how to construct the name of the definition representing an element declaration from the name of the element. The <i>name-pattern</i> must contain exactly one percent character (%). This character is replaced by the name of the element, after colon replacement, and the result is used as the name of the definition.
[input] inline-attlist / no-inline-attlist	<p>inline-attlist specifies not to generate definitions for attribute list declarations. Instead, attributes in attribute list declarations are moved into the definitions generated for element list declarations.</p> <p>no-inline-attlist generates a distinct definition (with combine="interleave") for each attribute list declaration in the DTD; each element declaration definition references the definition for the corresponding attribute list declaration.</p>

Table 58. DTD to XML (Trang) Document Wizard Fields

<i>Field</i>	<i>Description</i>
[input] attlist-define=<name-pattern>	Specifies how to construct the name of the definition representing an attribute list declaration from the name of the element. The name-pattern must contain exactly one percent character (%). This character is replaced by the element name, after colon replacement, and the result is used as the name of the definition.
[input] any-name=<name>	Specifies the name of the definition generated for the content of elements declared in the DTD as having a content model of ANY.
[input] strict-any	Preserves the exact semantics of ANY content models by using an explicit choice of references to all declared elements. By default, Trang uses a wildcard that allows any element.
[input] annotation-prefix=<prefix>	Default values are represented using an annotation attribute <i>prefix:defaultValue</i> where <i>prefix</i> is bound to http://relaxng.org/ns/compatibility/annotations/1.0 as defined by the RELAX NG DTD Compatibility Committee Specification. By default, Trang uses a for <i>prefix</i> unless that conflicts with a prefix used in the DTD.
[input] generate-start / no-generate-start	Specifies whether or not Trang should generate a start element. DTDs do not indicate what elements are allowed as document elements. Trang assumes that all elements that are defined but never referenced are allowed as document elements.
[output] encoding=<name>	Uses <i>name</i> as the encoding for output files.
[output] indent=<n>	Indents each indent level in the output file by <i>n</i> spaces.
[output] disable-abstract-elements	Disables the use of abstract elements and substitution groups in the generated XML schema.

Table 58. DTD to XML (Trang) Document Wizard Fields

<i>Field</i>	<i>Description</i>
[output] any-process-contents=strict lax skip	Specifies the value for the processContents attribute of any elements. The default is strict, which corresponds to DTD semantics.
[output] any-attribute-process-contents=strict lax skip	Specifies the value for the processContents attribute of anyAttribute elements. The default is skip, which corresponds to RELAX NG semantics.

◆ **To use the DTD to XML Schema (Trang) wizard:**

1. From the Stylus Studio menu bar, select **File > Document Wizards**.
The **Document Wizards** dialog box appears.
2. In the **XML Editor** tab, click **DTD to XML Schema (Trang)** and click **OK**.
The **DTD to XML Schema (Trang)** dialog box appears.

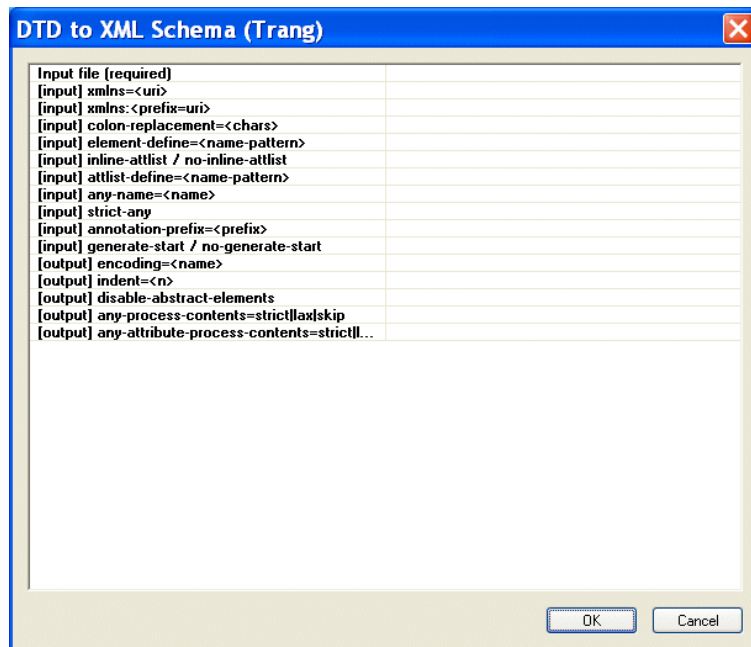


Figure 247. DTD to XML Schema (Trang) Dialog Box

3. Enter the absolute path of the DTD from which you want to create an XML Schema. The DTD must be encoded in UTF-8. You can type the file path, or use the browse button (which appears when you place the cursor in the **Input file (required)** field. This is the only required field.
4. Optionally, complete any of the remaining fields.
5. Click **OK**.
Stylus Studio displays the new XML Schema in the XML Schema Editor.

Creating XML Schema from an XML Document

There are two ways to create XML Schema from an XML document:

- The *XML to XML Schema document wizard* allows you to create XML Schema from any XML document. The XML document you use to create the XML Schema is not modified with attribute information about the new XML Schema when you use this method.
- The *Create Schema from XML Content feature* in the XML Editor. This method allows you to create an XML Schema (or DTD) based on the current XML document in the XML Editor. The XML document is always modified with namespace and schema location attributes when you use this method. If you choose to create DTD, you have the option of creating internal or external DTD.

Both methods allow you to specify the URI for the generated files (if an XML document has multiple namespaces defined for it, Stylus Studio creates a separate XML Schema associated with each namespace).

Using the XML to XML Schema Document Wizard

Use this procedure when you want to create an XML Schema based on the content of an existing XML document.

◆ To use the XML to XML Schema document wizard:

1. Select **File > Document Wizards** from the menu.
The **Document Wizards** dialog box appears.

2. Double-click **XML to XML Schema**.

The **Convert XML to XML Schema** dialog box appears.

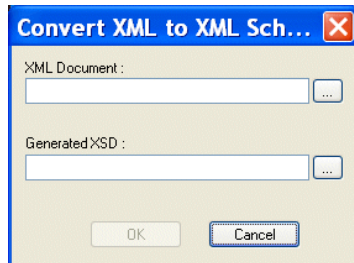


Figure 248. Convert XML to XML Schema Dialog Box

3. Specify the XML document you want to use to create an XML Schema in the **XML Document** field.
4. Specify the URI for the generated file(s) in the **Generated XSD** field.
5. Click the **OK** button.

Stylus Studio creates the XML Schema file and opens it in the XML Schema Editor.

Using the Create Schema from XML Content Feature

Use this procedure when you want to create an XML Schema based on the content of an existing XML document.

◆ **To use the Create Schema XML Content feature:**

1. Open the XML document from which you wish to create an XML Schema.

2. Select **XML > Create Schema from XML Content** from the Stylus Studio menu.
The **Create Schema or DTD** dialog box appears.

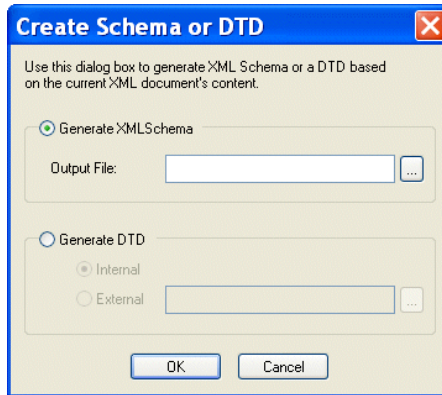



Figure 249. Create Schema or DTD Dialog Box

3. Click **Generate XML Schema**.
The **Output File** field becomes active.
4. Type a name for the XML Schema you want to create, or use the browse button () to search for an existing file.
5. Click the **Yes** button.
The XML Schema is created. If you do not specify a complete URL, the schema is written to the same location as the XML document from which it was created.

Displaying the New XML Schema

Use this procedure to open the new XML Schema created using the Create Schema from XML Content feature (or to open the XML Schema associated with any active XML document).

- ◆ **To display the new XML Schema:**
 1. Click **XML > Open Associated Schema**.
 2. Select the XML Schema from the drop-down menu.
The XML Schema appears in the XML Schema Editor.

Creating XML Schema from EDI



The EDI to XML Schema document wizards for EDIFACT, IATA, X-12, and EANCOM dialects are available only in Stylus Studio XML Enterprise Suite.

Stylus Studio provides several document wizards that allow you to create XML Schema based on EDI dialects like EDIFACT, X12, IATA, and EANCOM.

This section covers the following topics:

- “[Wizard Options](#)” on page 519
- “[Running an EDI to XML Schema Document Wizard](#)” on page 520

Wizard Options

Though specifics vary across EDI dialects (IATA and EANCOM refer to versions, while X12 refers to Release, for example), the options for most EDI document wizards are similar, if not the same. Document wizard options are summarized in the following table.

Table 59. EDI Document Wizard Options

<i>Option Name</i>	<i>Description</i>
Version	The version of the EDI dialect from which you want to create an XML Schema. For the X12 dialect, this option is referred to as Release.
Message	The specific message type from which you want to create an XML Schema. For the X12 dialect, this option is referred to as Transaction Set.
Include annotations describing each element	Whether or not you want to include annotations that describe each element in the generated XML Schema.
Generate enumerations for elements that have codelists	Whether or not you want the generated XML Schema to use enumerations for fields that have lists of values (<xsd:enumeration value="05">, <xsd:enumeration value="06">, and so on, for example).
Use long element names	Whether or not you want the generated XML Schema to use long or short element names (ACS04 versus ACS04-ShipmentMethodOfPayment, for example).

Table 59. EDI Document Wizard Options

<i>Option Name</i>	<i>Description</i>
Use “unbounded” for maxOccurs when loop value is 99 or higher	Whether or not you want to substitute “unbounded” for maxOccurs attributes that have a value of 99 or more. Stylus Studio bases maxOccurs values on the loops it detects in the EDI message or transaction.
Batch schema/ Interactive schema	Whether the EDI message or transaction is are being used in batch or interactive mode. This option does not pertain to the X12 dialect.

Running an EDI to XML Schema Document Wizard

◆ **To run an EDI to XML Schema documentation wizard:**

1. Select **File > Documentation Wizards** from the Stylus Studio menu.
The **Documentation Wizards** dialog box appears.

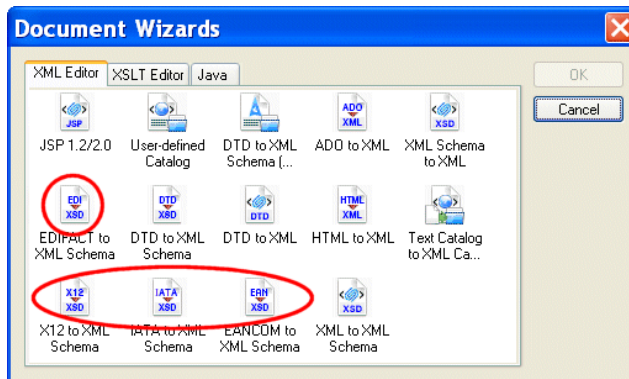


Figure 250. Document Wizards Dialog Box

2. Double-click the icon for the EDI document wizard you want to run.
A dialog box for the document wizard you select appears. The **Create XML Schema from EDIFACT Message Definition** dialog box is shown here.

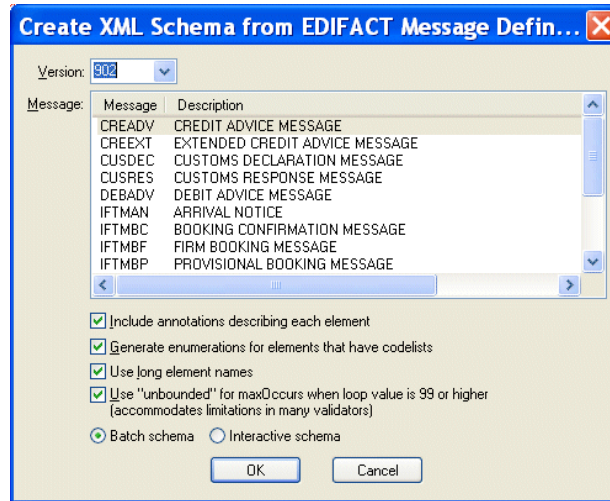


Figure 251. Create XML Schema from EDIFACT Message Definition

3. If necessary, change the version ID or release number. Values are listed chronologically in ascending order.
4. Select the message or transaction set on which you wish to base the XML Schema you are creating.
5. Change the XML Schema creation options as required. See [Wizard Options](#) on page 519 if you need help with this step.
6. Click OK.

Stylus Studio converts the EDI you selected in [Step 4](#) to XML Schema and opens a new, untitled document in the XML Schema Editor.

Working with XML Schema in Stylus Studio

You use the *XML Schema Editor* to view, define, and validate XML Schema using one or more of three tabs, or views. This section describes these and other tools for working with XML Schema in Stylus Studio.

This section covers the following topics:

- [Views in the XML Schema Editor](#) on page 523
- [Validating XML Schema](#) on page 526
- [Updating XML Schema Associated with a Document](#) on page 526
- [Viewing Sample XML](#) on page 526
- [Using XML Schema in XQuery and XSLT Mapper](#) on page 528
- [Printing](#) on page 528
- [Node Properties](#) on page 529

Views in the XML Schema Editor

The XML Schema Editor has **Diagram**, **Tree**, and **Documentation** tabs. The **Diagram** tab, which is the default for the XML Schema Editor, is shown in [Figure 252](#).

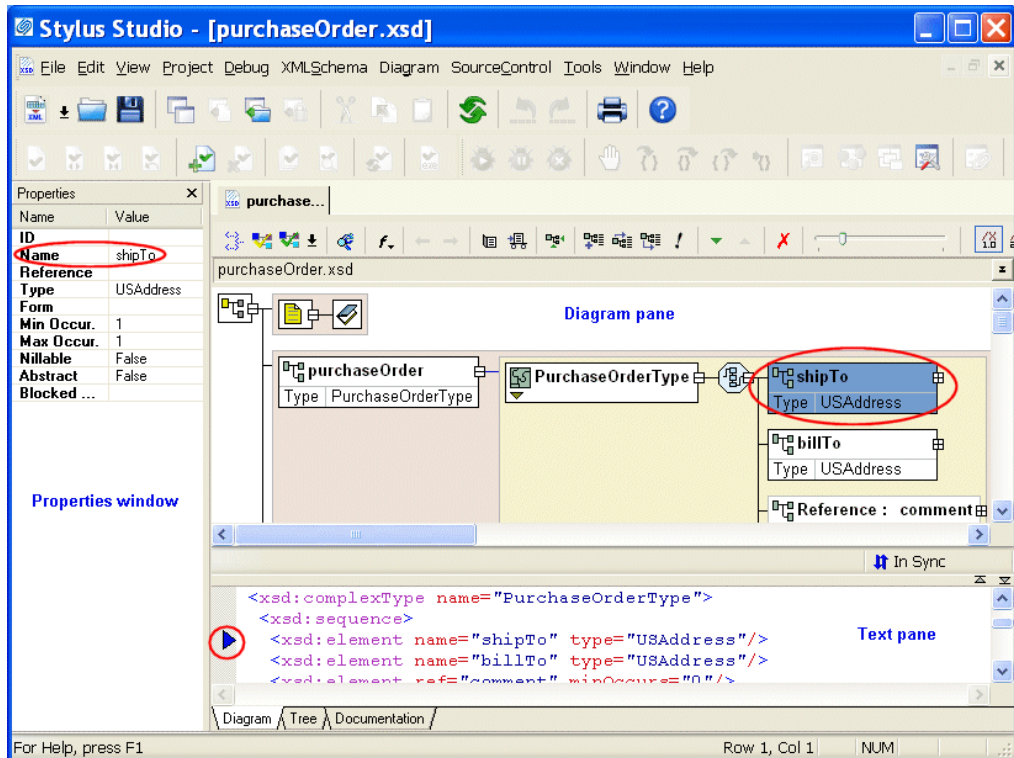


Figure 252. XML Schema Editor Diagram Tab



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the XML Schema Diagram Editor video](#).

You can see other Stylus Studio video demonstrations here:
http://www.stylusstudio.com/xml_videos.html.

Each tab displays the schema from a unique perspective, as summarized here:

- **Diagram** – Displays the XML Schema using graphical elements for the nodes (elements, attributes, simpleTypes, complexTypes, and so on) defined in the XML Schema in a *diagram pane*. Container elements can be expanded to show child

elements, and values such as element and attribute names and types can be edited in place by double-clicking the node you want to modify.

In addition to the diagram pane, the **Diagram** tab includes a *text pane*. The text pane displays the raw XML text used to define the XML Schema, and lets you see how the changes you make in the diagram affect the XML Schema text. You can make changes in either pane – to a node in the diagram, or directly to the text – Stylus Studio keeps both views synchronized.

The **Diagram** tab has a full complement of editing tools, including checkers for well-formedness and validation, as well as a query functionality that lets you evaluate your query using either XPath 1.0 or XPath 2.0. XML Schema query is also supported in the **Tree** tab. See “[Printing](#)” on page 528 to learn how to print information in the **Diagram** tab.

- **Tree** – Displays a DOM tree representation of the XML Schema. You can edit the XML Schema graphically, in the tree itself, or by modifying the properties of the nodes you select.

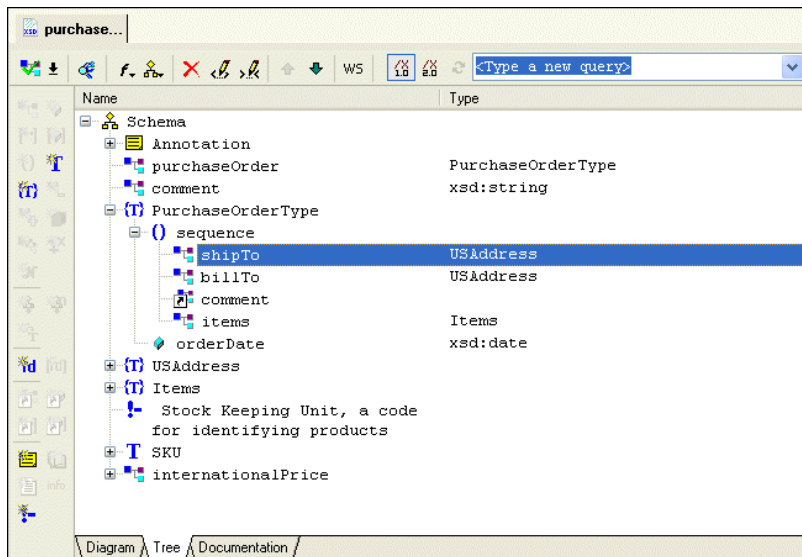


Figure 253. XML Schema Editor Tree Tab

- **Documentation** – Displays read-only summary and detailed reference information about the XML Schema, including sections for schema document properties, global declarations, and global definitions.

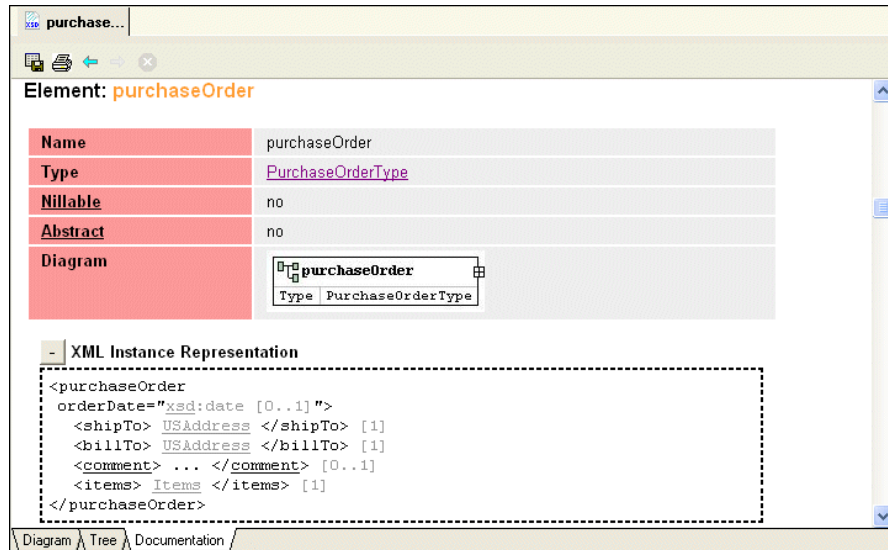



Figure 254. XML Schema Editor Documentation Tab (XS3P Format)

In Stylus Studio, the information in all tabs synchronized automatically.

Generally speaking, if you are just getting started with XML Schema you should consider using the **Diagram** tab to work with XML Schema in Stylus Studio. Its graphical user interface makes defining XML Schema easy and error-free, and the built-in text pane, which lets you see how new nodes are rendered in XML, can be a useful learning tool.

To get started using the **Diagram** view to define an XML Schema, see “[Defining an XML Schema Using the Diagram Tab – Getting Started](#)” on page 68. Procedures for working with the **Diagram** tab are also covered throughout this chapter.

Validating XML Schema

Stylus Studio can analyze your XML Schema document to determine if it is valid. At any time, click **Validate Document**  in the Stylus Studio tool bar to ensure that your schema is valid. If it is not, Stylus Studio displays a message that indicates the cause and location of the error. You can use any number of XML Schema validation engines to parse your XML Schema, including

- MSXML DOM Parser
- MSXML SAX Parser
- .NET XML Parser
- Built-in Java processor

Updating XML Schema Associated with a Document

Stylus Studio can associate an XML Schema with an XML document, and it can validate an XML document against its associated XML Schema. If you update an XML Schema in Stylus Studio and that schema is associated with an XML document that is open in Stylus Studio, Stylus Studio refreshes the XML Schema information for the XML document.

Viewing Sample XML

You can view a sample of the XML represented by a node in the XML Schema **Diagram** tab. You can also optionally create an XML document based on that instance. For

example, here is an instance of the XML represented by the purchaseOrder element in purchaseOrder.xsd.

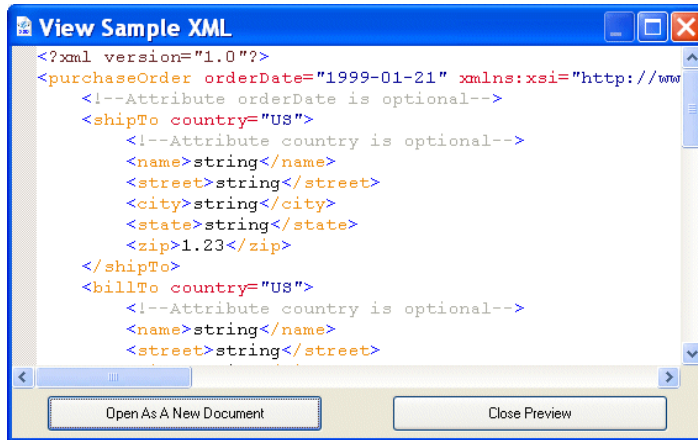


Figure 255. Sample of XML Based on XML Schema

If you want, you can create a new XML document based on this instance by clicking the **Open as New Document** button.

Note If the XML Schema contains an element defined using a built-in type, the instance of that element in the XML document is created using the minimum value of the range specified for that type. For example, if the XML Schema contains a `<part>` element defined as `type="xs:integer"`, the `<part>` element in the resulting XML document appears as `<part>-9223372036854775808</part>`.

◆ **To view sample XML:**

1. In the **Diagram** tab, select the node for which you want to see sample XML.
2. Select **Diagram > View XML Sample** from the Stylus Studio menu.
Alternative: Select **View XML Sample** from the node's shortcut menu (right-click to display).
The **View Sample XML** dialog box appears. (See [Figure 255](#).)
3. If you want to open this instance as a new XML document in Stylus Studio, click the **Open as a New Document** button. Otherwise, click **Close Preview**.

Using XML Schema in XQuery and XSLT Mapper

You can use an XML Schema as the source document or target document for Stylus Studio's XQuery and XSLT Mappers. See [“Building an XQuery Using the Mapper”](#) on page 750 and [“Creating XSLT Using the XSLT Mapper”](#) on page 449 for more information on these topics.

Printing


You can print XML Schema from the **Diagram** tab, and you can print XML Schema documentation from the **Documentation** tab.

Printing XML Schema

Stylus Studio allows you to print either the graphics in the diagram pane, or the raw XML in the text pane. If one pane is collapsed, Stylus Studio prints the visible pane. If both panes are visible, Stylus Studio prints the pane that currently has focus.

Tip Select **File > Print Preview** to verify the output before you print.

◆ **To print XML Schema from the Diagram tab:**

1. Select the pane of the **Diagram** tab you want to print.
2. Click **Print** .

Alternative: Select **File > Print** from the Stylus Studio menu.

Printing XML Schema Documentation

To print XML Schema documentation, click the **Print** tool (or Ctrl + P) on the **Documentation** tab. Stylus Studio prints the XML Schema documentation using the XS3P format. See [“Generating Documentation for XML Schema”](#) on page 580 for more information.

Node Properties

The **Properties** window is available when you are using the **Diagram** or **Tree** tab of the XML Schema Editor. When the **Properties** window is open, it displays the properties of the node you click. If you have selected a restricted node from a redefined XML Schema, Stylus Studio displays a separate section in the lower half of the **Properties** window for you to specify the facets, as shown in [Figure 256](#).

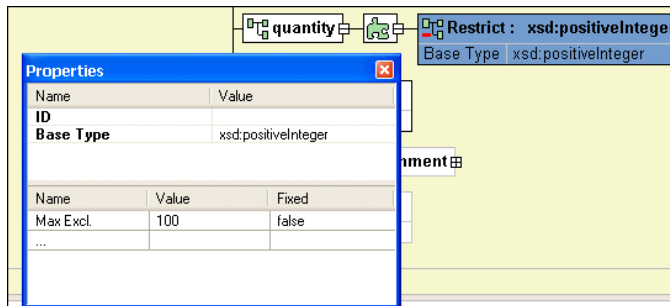


Figure 256. Properties Window with Restricted Type Facets

If the **Properties** window is not visible, select **View > Properties** from the Stylus Studio menu.

Tip The **Properties** window is a dockable window – you can drag it out of Stylus Studio and place it anywhere on your desktop, as shown in [Figure 256](#).

To change the value of a property, click the property field and enter the new value. If only certain values are allowed, Stylus Studio displays a drop-down list of the valid choices. Each type of node has its own set of properties. For a description of each property, see [“About XML Schema Properties”](#) on page 587.

Working with Properties in the Diagram

You can also display and edit properties within the nodes in the diagram. See [“Displaying Properties”](#) on page 72 for more information on this feature.

Getting Started with XML Schema in the Tree View

This section provides a quick tour of the main features of the **Tree** view in the XML Schema Editor. It provides instructions that you can follow to define a simple XML Schema.

This section provides step-by-step instructions for defining the bookstoreTree.xsd XML Schema document. You should perform the steps in each topic in the order of the topics. This section covers the following topics:

- [Description of Sample XML Schema](#) on page 530
- [Defining a complexType in a Sample XML Schema in the Tree View](#) on page 531
- [Defining Elements of the Sample complexType in the Tree View](#) on page 535

For instructions for using the **Diagram** view to define the same XML Schema, see [“Defining an XML Schema Using the Diagram Tab – Getting Started”](#) on page 68.

Description of Sample XML Schema

Suppose you want to define an XML Schema that defines book, magazine, and newsletter elements. The type of each of these elements is `PublicationType`. The XML Schema defines the `PublicationType` complexType. An element that is a `PublicationType` contains the following:

- The `genre` attribute specifies the style of the publication.
- There is always exactly one `title` element.
- The `subtitle` element is optional.
- There must be at least one `author` element and there can be more. Each `author` element contains one `first-name` element and one `last-name` element.
- Of the following three elements, exactly one must always be present:
 - `ISBNnumber`
 - `PUBnumber`
 - `LOCnumber`
- The elements must be in the order specified in this list.

Tips for Adding Nodes

To add a node to an XML Schema in the **Tree** view, click a node that is already in the schema. Stylus Studio activates the buttons for only those nodes that can be children of the node you selected. If you hold down the Shift key, Stylus Studio activates only those buttons that allow you to add nodes that can be siblings of the selected node.




Defining a complexType in a Sample XML Schema in the Tree View

The steps for defining the `PublicationType` complexType are described in the following sections:

- [Defining the Name of the Sample complexType in the Tree View](#) on page 531
- [Adding an Attribute to a Sample complexType in the Tree View](#) on page 532
- [Adding Elements to a Sample complexType in the Tree View](#) on page 532
- [Adding Optional Elements to a Sample complexType in the Tree View](#) on page 533
- [Adding an Element That Contains Subelements to a complexType in the Tree View](#) on page 533
- [Choosing the Element to Include in the Sample complexType in the Tree View](#) on page 534

Defining the Name of the Sample complexType in the Tree View


◆ To define a complexType in the sample XML Schema:

1. From the Stylus Studio menu bar, select **File > New > XML Schema**.
Stylus Studio displays the XML Schema Editor.
2. At the bottom of the XML Schema Editor, click the **Tree** tab.
Stylus Studio displays the **Tree** view of the schema, and the **Properties** window, which lists the properties for the selected node in the tree.
3. Click the **Schema** node .
4. In the left tool bar, click **New complexType** .
5. Type `PublicationType` as the name for this new complexType and press Enter.
6. Click **Save** .

7. In the **Save** dialog box that appears, in the **URL** field, type `bookstoreTree.xsd`, and click **Save**.



Adding an Attribute to a Sample complexType in the Tree View

◆ **To add the genre attribute to the PublicationType complexType:**

1. In the **Tree** view, click the **PublicationType** node.
2. In the left tool bar, click **New Attribute Definition** .
In the **Tree** view, Stylus Studio displays a field for the new attribute.
3. Type `genre` as the name of the new attribute and press Enter.
Stylus Studio displays a drop-down list of built-in simpleTypes.
4. Double-click `xsd:string`.


Adding Elements to a Sample complexType in the Tree View

◆ **To add the title element, which must appear exactly once, to the PublicationType complexType:**

1. In the **Tree** view, click the **PublicationType** node.
2. In the left tool bar, click **New Model Group** .
Stylus Studio displays a drop-down list of group modifiers.
3. Double-click the **sequence** modifier.
The **sequence** modifier indicates that an instance document contains zero, one, or more of each child element in the order in which they are defined.
4. In the left tool bar, click **New Element Definition** .
In the **Tree** view, Stylus Studio displays a field for the new element definition.
5. Type `title` as the name of the new element and press Enter.
Stylus Studio displays a drop-down list of built-in simpleTypes.
6. Double-click `xsd:string`.

Adding Optional Elements to a Sample complexType in the Tree View

◆ To add the optional subtitle element to the `PublicationType` complexType:




1. In the **Tree** view, click the **sequence** node.
2. In the left tool bar, click **New Element Definition** .
In the **Tree** view, Stylus Studio displays a field for the new element definition.
3. Type `subtitle` as the name of the new element and press Enter.
Stylus Studio displays a drop-down list of built-in simpleTypes.
4. Double-click `xsd:string`.
5. In the **Properties** window, double-click the **Min Occur.** field.
6. Type 0 and press Enter
7. In the **Properties** window, double-click the **Max Occur.** field.
8. Type 1 and press Enter.


Adding an Element That Contains Subelements to a complexType in the Tree View

The `PublicationType` complexType must include at least one author element. An author element must include a first-name element and a last-name element.

Each element that can contain subelements is a complexType. Consequently, to add the author element to the `PublicationType` complexType, you must first define the `authorType` complexType. You can then add an element that is of `authorType` to the `PublicationType` complexType.



◆ To define the `authorType` complexType:


1. In the XML Schema Editor, click the Schema node.
2. In the left tool bar, click **New complexType** .
Stylus Studio displays a field for the new complexType.
3. Type `authorType` as the name for this new complexType and press Enter.
4. In the left tool bar, click **New Model Group** .
5. In the drop-down list that appears, double-click the **sequence** modifier.
6. In the left tool bar, click **New Element Definition** .

7. In the field that Stylus Studio displays, type **first-name** as the name of the new element and press Enter.
 8. In the drop-down list that appears, double-click **xsd:string**.
 9. In the **Tree** view, click the **sequence** modifier for authorType.
 10. Repeat [Step 6](#) through [Step 8](#), but type last-name as the name of the new element.
- ◆ **Now you can add the author element to the PublicationType complexType:**
1. In the **Tree** view, click the **sequence** node under the **PublicationType** node.
 2. In the left tool bar, click **New Element Definition** .
 3. In the field that Stylus Studio displays, type **author** as the name of the new element and press Enter.
 4. In the drop-down list that appears, double-click **authorType**.
 5. In the **Properties** window, double-click in the **Min Occur.** field.
 6. Type 1 and press Enter.
 7. In the **Properties** window, double-click in the **Max Occur.** field.
 8. Type unbounded in the **Max Occur.** field and press Enter.

Choosing the Element to Include in the Sample complexType in the Tree View

In the sample XML Schema, you want PublicationType elements to contain an ISBNnumber, PUBnumber, or LOCnumber element.



- ◆ **To specify this:**
1. In the **Tree** view, under the **PublicationType** node, click the **sequence** node.
 2. In the left tool bar, click **New Model Group** .
 3. In the drop-down list that appears, double-click the **choice** modifier.
 4. In the left tool bar, click **New Element Definition** .
 5. In the field that Stylus Studio displays, type ISBNnumber as the name of the new element and press Enter.
 6. In the drop-down list that appears, scroll until you can double-click **xsd:integer**, or type xsd:integer and press Enter.

7. In the **Tree** view, click the **choice** modifier for `PublicationType`.
8. Repeat [Step 4](#) through [Step 7](#) two more times. Once for the `PUBnumber` element and once for the `LOCnumber` element.
9. Click **Save** .

Stylus Studio displays a message that indicates that the schema is not valid. Click **OK** in the error box. In the **Output Window**, you can see the following message:

```
Validating bookstoreTree.xsd...
file:///c:/yourDirectory/bookstoreTree.xsd:6,45:
Invalid child 'sequence' in the complexType
The XML document bookstoreTree.xsd is NOT valid (1 errors)
```



The problem is that there is a sequence element after an attribute element. The sequence element must come first.

10. Right-click the **genre** node.
11. In the pop-up menu that appears, click **Move Down**.
12. Click **Validate Document**  and **Save** .

The definition of the `PublicationType` complexType is now complete and the schema is now valid.

Defining Elements of the Sample complexType in the Tree View

◆ To define the `book`, `magazine`, and `newsletter` elements in the sample XML Schema:

1. In the **Tree** tab, click the **Schema** node.
2. In the left tool bar, click **New Element Definition** .
3. In the field that Stylus Studio displays, type **book** as the name of the new element and press Enter.
4. In the drop-down list that appears, double-click **PublicationType**.
5. Repeat [Step 1](#) through [Step 4](#) two more times. Once for the `magazine` element and once for the `newsletter` element.
6. Click **Save** .

This is the end of the section that provides instructions for getting started with defining XML Schemas in the **Tree** view. The topics that follow this topic provide complete

information for defining the elements that can be in an XML Schema and for working in the **Diagram**, **Tree**, and **Text** views.

Defining simpleTypes in XML Schemas

Many simpleTypes, such as `string` and `integer`, are built in to an XML Schema. You can define your own simpleType by restricting the range of values provided by a built-in simpleType. You can also define simpleTypes that are derived from the simpleTypes you define.

This section covers the following topics:

- [About simpleTypes in XML Schemas](#) on page 536
- [Examples of simpleTypes in an XML Schema](#) on page 537
- [Defining a simpleType in the Diagram View](#) on page 538
- [Defining a simpleType in the Tree View](#) on page 542
- [About Facet Types for simpleTypes](#) on page 544
- [Defining List and Union simpleTypes in the Tree View](#) on page 545

About simpleTypes in XML Schemas

XML Schema defines several kinds of simpleTypes:

- *Atomic* is the term for most of the simpleTypes built in to XML Schema, such as `integer`, `string`, and `decimal`. They are called “atomic” because in the context of XML Schema, no part of an element or attribute of an atomic type has meaning on its own. It is only the whole instance that has meaning.

Descriptions of the XML Schema built-in types are in the *W3C XML Schema Part 2: Datatypes* at <http://www.w3.org/TR/xmlschema-2/>.

- *List* simpleTypes are sequences of atomic types. All elements of a particular list simpleType are instances of the same atomic type.
- *Union* simpleTypes are sequences of atomic and list types. However, the elements of a particular union simpleType can be instances of more than one atomic or list type.
- *Anonymous* simpleTypes are simpleType definitions that are not explicitly named. An anonymous simpleType can be an atomic, list, or union simpleType. You define an anonymous simpleType in the element or attribute definition that uses it.

Anonymous simpleTypes are useful when you want to define a type that is used in only one element or attribute. By specifying an anonymous simpleType, you save the overhead of explicitly defining the type and specifying a reference to it.

Examples of simpleTypes in an XML Schema

The *W3C XML Schema Part 0: Primer* specifies the following simpleType in its sample purchase order schema:

```
<!-- Stock Keeping Unit, a code for identifying products -->
<xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>
```

This specifies that SKU is a simpleType. It is restricted to the values of the base type, which is `xsd:string`. This means that for a node that is of type SKU, the possible values are a subset of the values allowed for the `xsd:string` type.

The `xsd:pattern` element specifies that the pattern facet is being applied to the set of values allowed by the `xsd:string` type. The value of the `xsd:pattern` element is an XML Schema regular expression that specifies the allowable values for nodes of type SKU. In this example, the regular expression specifies that the value must be three digits, followed by a hyphen, followed by two uppercase ASCII letters – `<xsd:pattern value="\d{3}-[A-Z]{2}"/>`. For information about XML Schema expressions, see the [W3C XML Schema Part 0: Primer](http://www.w3.org/TR/xmlschema-0/) at <http://www.w3.org/TR/xmlschema-0/>.

Elsewhere in the purchase order schema, an attribute definition specifies that SKU is the type of its value:

```
<xsd:attribute name="partNum" type="SKU" use="required"/>
```

An XML document that uses a schema that contains this simpleType definition can specify the `partNum` attribute. The parser ensures that the value of the `partNum` attribute is in the range specified by the `xsd:pattern` element. The SKU type itself is not mentioned in the instance document.

Following is another example of a simpleType definition from the *W3C XML Schema Part 0: Primer*. This simpleType, `myInteger`, is based on the `xsd:integer` type. It specifies two

facets (`minInclusive` and `maxInclusive`), which specify the lower and upper inclusive bounds of the range of valid values.

```
<xsd:simpleType name="myInteger">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="10000"/>
    <xsd:maxInclusive value="99999"/>
  </xsd:restriction>
</xsd:simpleType>
```

Defining a simpleType in the Diagram View

This section describes the procedures for defining simpleTypes in the **Diagram** view. It covers the following topics:

- [“Before You Begin”](#) on page 538
- [“Defining an Atomic simpleType”](#) on page 538
- [“Specifying a Restriction for a simpleType – QuickEdit”](#) on page 539
- [“Specifying a Restriction for a simpleType – Manually”](#) on page 540
- [“Defining List and Union simpleTypes”](#) on page 541


Before You Begin

Many of the editing features used in this section are described in [“Defining an XML Schema Using the Diagram Tab – Getting Started”](#) on page 68. You should familiarize yourself with that material if you have not done so already.

Defining an Atomic simpleType

This topic provides the steps for defining an atomic simpleType in the **Diagram** view.

◆ **In the Diagram view, to define an atomic simpleType:**

1. Right-click the schema node  to display the shortcut menu.
2. Select **Add > simpleType**.
The new simpleType appears in the diagram; its properties are displayed in the **Properties** window.
3. Change the default name to the name of the new simpleType and press Enter.

Specifying a Restriction for a simpleType – QuickEdit

QuickEdit is a feature that combines commonly-performed editing operations, such as specifying a restriction for a simpleType. You can also perform this operation in a different way. See “[Specifying a Restriction for a simpleType – Manually](#)” on page 540.

◆ To specify a restriction for a simpleType using QuickEdit:

1. Right-click the simpleType node  to display the shortcut menu.
2. Select **QuickEdit > Derive by restriction** from the shortcut menu.

The **Type Derivation** dialog box appears.

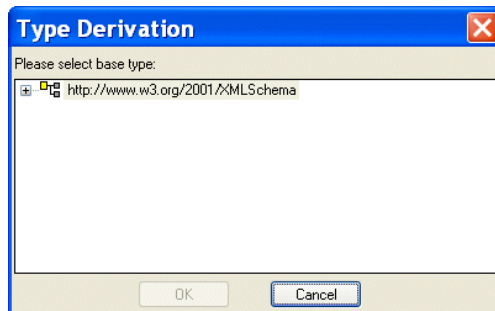


Figure 257. Type Derivation Dialog Box

The **Type Derivation** dialog box displays the W3C XML Schema, as well as any referenced XML Schemas.

3. Expand the schema (click the plus sign) to display the base types associated with that XML Schema.
4. Select the type on which you wish to base the simpleType you are defining and click **OK**.

The simpleType is updated with an element that identifies the restricted type:

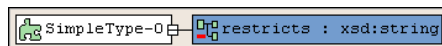


Figure 258. Restricted Type

In the lower half of the **Properties** window, Stylus Studio displays a section that allows you to specify facets – values that define the constraint on the range of values allowed by the base type.

5. Click the **Name** field and select a facet type.

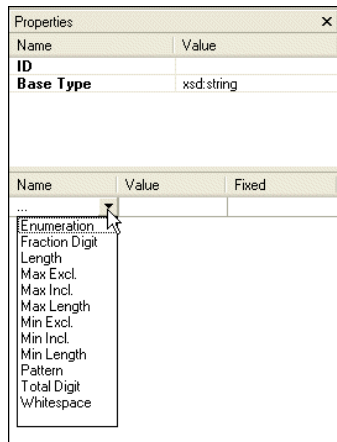


Figure 259. Specifying Facets for a Restricted Type

Stylus Studio displays only those facets that are allowed for the base type you selected. For a description of each facet, see [“About Facet Types for simpleTypes”](#) on page 544.

6. Click the **Value** field for a facet you want to specify.
7. Enter a value for the new facet.
8. To specify another facet, repeat [Step 5](#) through [Step 7](#).

Specifying a Restriction for a simpleType – Manually

This procedure describes how to specify a restriction for a simpleType manually. It is an alternative to the procedure described in [“Specifying a Restriction for a simpleType – QuickEdit”](#) on page 539.

- ◆ **To specify a restriction for a simpleType manually:**
 1. Right-click the simpleType node to display the shortcut menu.
 2. Select **Add > Restriction** from the shortcut menu.

The simpleType is updated with a restriction icon:

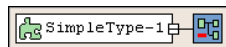


Figure 260. Restriction Icon

3. Select the restriction icon if it is not already selected.
4. In the **Properties** window, select the type on which you want to base the simpleType you are defining from the **Base Type** field.

At the bottom of the **Properties** window, Stylus Studio displays a section that allows you to specify facets – values that define the constraint on the range of values provided by the base type.

5. Click the **Name** field and select a facet type.

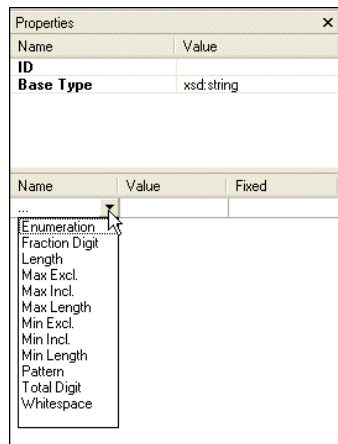


Figure 261. Specifying Facets for a Restricted Type

Stylus Studio displays only those facets that are allowed for the base type you selected. For a description of each facet, see [“About Facet Types for simpleTypes”](#) on page 544.

6. Click the **Value** field for a facet you want to specify.
7. Enter a value for the new facet.
8. To specify another facet, repeat [Step 5](#) through [Step 7](#).

Defining List and Union simpleTypes

The procedure for defining list and union simple types is similar:

1. Create the simpleType as described in [“Defining a simpleType in the Diagram View”](#) on page 538.

2. Select the type (list or union) from the shortcut menu (right-click the new `simpleType` and select **Add > List** or **Add > Union**).
3. Specify the nodes that comprise the `simpleType`'s list or union. These types are restricted to annotations and other `simpleTypes`.

How you perform this last step depends on whether you are adding new or existing annotation or `simpleType` nodes to the list or union.

- To add a new, undefined annotation or `simpleType` to the list or union, right click the list or union and select **Add > Annotation** or **Add > SimpleType** from the shortcut menu.
- To add an existing annotation or `simpleType` to the list or union, drag the annotation or `simpleType` to the list or union, and drop it there, as shown in [Figure 262](#), which shows `SimpleType-6` being added to the union `SimpleType-3`.

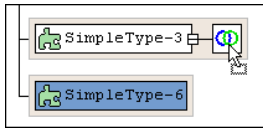


Figure 262. Dragging Nodes to Define Other Nodes

Notice that the pointer changes shape when you place it over an appropriate target node.


Defining a `simpleType` in the Tree View

This topic provides the steps for defining an atomic `simpleType` in the **Tree** view. When you are familiar with this procedure, you can adapt it to define list, union, and anonymous `simpleTypes`.


◆ In the **Tree view**, to define an atomic `simpleType`:

1. Click the node you want to define a `simpleType` for. This can be one of the following types of nodes:
 - schema
 - element
 - attribute
 - list
 - union

To define a simpleType as the sibling of another node, click the node and hold down the Shift key when you click the button in [Step 2](#). You cannot, of course, define a simpleType as a sibling of the **Schema** node.


2. In the left tool bar, click **New simpleType** . Stylus Studio displays an empty simpleType field as the last child of the node you selected. If you held down the Shift key, the field is the last sibling of the selected node.

3. Type a name for the new simpleType and press Enter.

4. In the left tool bar, click **New Restriction** . A restriction specifies the type that the new simpleType is derived from. This is the base type.

Stylus Studio displays a scrollable list of XML Schema built-in types, and any simpleTypes you already defined in this schema. Descriptions of the XML Schema built-in types are in the [W3C XML Schema Part 0: Primer](#) at <http://www.w3.org/TR/xmlschema-0/>.

5. Double-click the simpleType that you want to base your new simpleType on.

6. In the left tool bar, click **New Facet** . A facet specifies a constraint on the range of values provided by the base type. Stylus Studio displays a scrollable list of XML Schema facet types. For a description of each facet, see [“About Facet Types for simpleTypes”](#) on page 544.

You must ensure that you specify a facet that is valid for the specified base type. Stylus Studio does not prevent you from specifying an invalid facet. The *W3C XML Schema Part: 0 Primer* includes a [table](#) at <http://www.w3.org/TR/xmlschema-0/> that provides this information.

7. Double-click the type of facet you want to specify.

8. In the **Properties** window, double-click the **Value** field.

9. Enter a value for the new facet.

10. To add another facet, click the **restriction** node for your simpleType, and repeat [Step 6](#) through [Step 9](#).

About Facet Types for simpleTypes

Table 60 provides a brief description of what you should specify as the value of a facet for a new simpleType. You should consult the [XML Schema Recommendation](#) for a complete definition of each facet and its allowable values.

Table 60. Facet Values for simpleTypes

<i>Facet</i>	<i>Value</i>
enumeration	One allowable value. Add an enumeration facet for each allowable value. For example: <pre><xsd:simpleType name="USState"> <xsd:restriction base="xsd:string"> <xsd:enumeration value="AK"/> <xsd:enumeration value="AL"/> <xsd:enumeration value="AR"/> <!-- and so on ... --> </xsd:restriction> </xsd:simpleType></pre>
fractionDigits	The maximum number of digits that are allowed in the fractional portion of values of simpleTypes that are derived from xsd:decimal.
length	The number of units of length. Units vary according to the base type. The simpleType must be this number of units of length. For example, if xsd:string is the base type, you might specify 5 as the length if you know that each value will be a code that always has five characters.
maxExclusive	The exclusive upper bound of the range of values allowed for this simpleType. The value of the simpleType must be less than the value of maxExclusive.
maxInclusive	The inclusive upper bound of the range of values allowed for this simpleType. The value of the simpleType must be less than or equal to the value of maxInclusive.
maxLength	The maximum number of units of length. Units vary according to the base type. The length of the instances of this simpleType must be less than or equal to this number of lengths.



Table 60. Facet Values for simpleTypes

<i>Facet</i>	<i>Value</i>
minExclusive	The exclusive lower bound of the range of values allowed for this simpleType. The value of the simpleType must be more than the value of minExclusive.
minInclusive	The inclusive lower bound of the range of values allowed for this simpleType. The value of the simpleType must be equal to or more than the value of minInclusive.
minLength	The minimum number of units of length. Units vary according to the base type. The length of the instances of this simpleType must be equal to or more than this number of lengths.
pattern	A regular expression. The values of the simpleType must be literals that match this regular expression.
totalDigits	The maximum number of digits that are allowed in values of simpleTypes that are derived from xsd:decimal.
whiteSpace	Specify one of the following values: <ul style="list-style-type: none"> ● preserve indicates that no normalization is done. The value is not changed. ● replace indicates that each tab, line feed, and return is replaced with a space. ● collapse indicates that the processing specified by replace is done, and then contiguous sequences of spaces are collapsed into one space.

Defining List and Union simpleTypes in the Tree View

Sometimes you need to define a simpleType for a sequence of atomic types. In a list simpleType, all instances in the sequence must be of the same type. In a union simpleType, the instances in the sequence can be of different types. The procedure for defining list and union simpleTypes is the same.

◆ **In the Tree view, to define a list or union simpleType:**

1. Click the node you want to define the list or union type for.
2. In the left tool bar, click **New simpleType** . Stylus Studio displays an empty simpleType field as the last child of the node you selected.
3. Type a name for the new simpleType and press Enter.
4. In the left tool bar, click **New Aggregator** . Stylus Studio displays a pop-up menu with two choices.
5. Double-click **list** or **union**.
6. Define the atomic simpleType of the elements or attributes that are instances of the list or union type. See “[Defining simpleTypes in XML Schemas](#)” on page 536.
7. If you are defining a list type you are done. If you are defining a union type, click the **union** node and define another atomic simpleType that can be in the union. Perform this step for each type in the union.

Defining complexTypes in XML Schemas

In an XML Schema, an element that contains only data is a simpleType. Elements with any other contents are complexTypes. (Attributes are always simpleTypes.) The *XML Schema Recommendation* does not include any built-in complexTypes. You must define each complexType you need.

In the **Diagram** view, when you define a complexType as a top-level definition, it is a global declaration. You can specify that any element in the schema is of this complexType. Similarly, in the **Tree** view, it is a global declaration when you define a complexType as a child of the Schema node.

Tip Define the complexType first. Then when you define an element, Stylus Studio includes your complexType’s name in the menu that lists the available types for your new element. You can select the name of the complexType from the menu.

You can also define a complexType in the definition of an element. See “[Defining Elements That Contain Subelements in XML Schemas](#)” on page 560.

Stylus Studio takes care of most of the details for you. But as you define a complexType, it is helpful to keep in mind that a complexType node can have only one child node that is a model group modifier. However, this modifier node can have any number of child nodes that are modifiers. In this way, you can specify any number of modifiers in a


complexType. Each modifier controls the occurrence of its child nodes. You can specify the same modifier more than once. For example, you might want to specify the sequence modifier, with some child nodes, then the choice modifier with some child nodes, and then the sequence modifier again with other child nodes.


This section discusses the following topics:

- [Defining complexTypes That Contain Elements and Attributes – Diagram View](#) on page 547
- [Defining complexTypes That Contain Elements and Attributes – Tree View](#) on page 551
- [Defining complexTypes That Mix Data and Elements](#) on page 553
- [Defining complexTypes That Contain Only Attributes](#) on page 555

Defining complexTypes That Contain Elements and Attributes – Diagram View

◆ To define a complexType in the Diagram view:

1. Right-click the schema node .
2. In the shortcut menu, select **Add > ComplexType**.

Alternatives: This operation is also available from the **XMLSchema > Diagram > Add > ComplexType** menu and from the **Add** button .

The new complexType is added to the XML Schema. It is displayed in the diagram and in the text pane (if you have it open). The properties for the new complexType are displayed in the **Properties** window.

Adding Nodes to a complexType

Once you have created a complexType, you can further define it by adding sequences, elements, and other nodes. The basic procedure for adding nodes to a complexType is to:

1. Select the node.
2. Use the menus or tool bar to add the node.
3. Fully describe the complexType and its nodes by editing values in the **Properties** window.

You can use this procedure to add the following nodes to a complexType:

- all
- annotation
- anyAttribute
- attribute
- attributeGroup
- choice
- group
- sequence


Next steps vary according to the constraints on the elements in the complexType. The following instructions show how to achieve some typical constraints.

Choosing an Element


Suppose you want to define a complexType that contains exactly one element, and that element can be one of several different elements. In XML Schema, you do this by defining `xsd:choice`.

◆ To define `xsd:choice` in the Diagram tab:

1. Right-click the icon that represents your new complexType.
2. In the shortcut menu that appears, select **Add > Choice**.

Alternatives: This operation is also available from the **XMLSchema > Diagram > Add > Choice** menu and from the **Add** button .

Stylus Studio displays the choice icon  alongside the complexType icon.

3. Right-click the choice icon and select **Add > Element**, or use the **Add** button . A element is added to the choice icon.

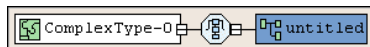





Figure 263. Defining Choice for a complexType

4. Make sure the new element is selected. In the **Properties** window, click the **Type** field.
5. Enter or select the type of the element.
6. Repeat [Step 3](#) through [Step 5](#) for each element that might be in the complexType.

Including All Elements

Suppose you want to define a complexType that contains a number of elements, the elements can be in any order, and there must be zero or one of each element. In XML Schema, you do this by defining `xsd:all`.


◆ To define `xsd:all` in the Diagram tab:

1. Right-click the icon that represents your new complexType.
2. In the shortcut menu that appears, select **Add > All**.
Alternatives: This operation is also available from the **XMLSchema > Diagram > Add > All** menu and from the **Add** button .
3. Stylus Studio displays the `all` icon  alongside the complexType icon.
Right-click the choice icon and select **Add > Element**, or use the **Add** button . A element is added to the `all` icon.
4. Make sure the new element is selected. In the **Properties** window, click the **Type** field.
5. Enter or select the type of the element.
6. If the element is required, go to [Step 7](#). If the element is optional, click the **Min Occur.** field in the **Properties** window, and type a zero (0).
7. If there must always be exactly one of this element, go to [Step 8](#). If there can be more than one of this element, click the **Max Occur.** field in the **Properties** window, and enter the maximum number allowed or click **unbounded** in the drop-down list.
8. Repeat [Step 3](#) through [Step 7](#) for each element that can be in the complexType.


Specifying the Sequence of Elements

Suppose you want to define a complexType that contains a number of elements in a particular order. The default is that each element must appear exactly once. However, some elements are optional, and some elements can appear more than once. In XML Schema, you do this by defining `xsd:sequence`.

◆ To define `xsd:sequence` in the Diagram tab:

1. Right-click the icon that represents your new complexType.
2. In the shortcut menu that appears, select **Add > Sequence**.
Alternatives: This operation is also available from the **XMLSchema > Diagram > Add > Sequence** menu and from the **Add** button .

Stylus Studio displays the sequence icon  alongside the complexType icon.

3. Right-click the sequence icon and select **Add > Element**, or use the **Add** button . A element is added to the sequence icon.
4. Make sure the new element is selected. In the **Properties** window, click the **Type** field.
5. Enter or select the type of the element.
6. If the element is required, go to [Step 7](#). If the element is optional, click the **Min Occur.** field in the **Properties** window, and type a zero (0).
7. If there must always be exactly one of this element, go to [Step 8](#). If there can be more than one of this element, click the **Max Occur.** field in the **Properties** window, and enter the maximum number allowed or click **unbounded** in the drop-down list.
8. Repeat [Step 3](#) through [Step 7](#) for each element that can be in the complexType.

Reordering Nodes

If you make a mistake in the order in which you specify nodes in your XML Schema (when specifying elements in a sequence, for example), you can rearrange them.

◆ **To reorder nodes in the diagram view:**

1. Click the node you want to move.
2. Click the **Move Up**  or **Move Down**  from the Stylus Studio tool bar until the node is positioned where you want it.

Alternative: This operation is also available from the **XMLSchema** menu and from the node's shortcut menu.

Combining the Sequence and Choice Modifiers



Suppose you want to define a complexType that contains a number of elements in a particular order, but some of them are optional, and you want to ensure that only one element from a particular group of elements is present. In other words, you need to combine the use of the sequence and choice modifiers. To define this, you must define a sequence modifier first. You can then define sequence and choice modifiers that are children of the initial sequence modifier.

Defining complexTypes That Contain Elements and Attributes – Tree View


The purchaseOrder.xsd sample document contains the following complexType definition. This complexType defines three elements, refers to a fourth element, and defines an attribute.






```
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items" type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

◆ In the Tree view, to define a complexType with a similar structure:

1. Click the **Schema** node.
2. In the left tool bar, click **New complexType** . In the **Tree** view, Stylus Studio displays a field for the new complexType.
3. Type a name for this new complexType and press Enter.
4. In the left tool bar, click **New Model Group** . Stylus Studio displays a pop-up menu that lists the group modifiers.
5. Double-click the modifier you want. For a description of each modifier, see [“Model Group Properties in XML Schemas”](#) on page 594.

You can specify any number of modifiers in a complexType. Each modifier controls the occurrence of its child nodes. You can specify the same modifier more than once. For example, you might want to specify the sequence modifier, with some child nodes, then the choice modifier with some child nodes, and then the sequence modifier again with other child nodes.

6. For each element that you want to define in this complexType with the selected modifier, perform the following steps:
 - a. Click the modifier name in the **Tree** view.
 - b. In the left tool bar, click **New Element Definition** . In the **Tree** view, Stylus Studio displays a field for the new element definition.

- c. Type a name for the new element and press Enter. Stylus Studio displays a pop-up menu that lists the built-in simpleTypes and simpleTypes you defined.
 - d. Double-click the type for the new element.
 - e. In the **Properties** window, you can double-click the field for any property to set the value for that property.
For example, you can specify 0 for the **Min Occur.** property and 1 for the **Max** property. The effect is that the element is optional.
7. For each element or group that you want to refer to in this complexType with the selected modifier, perform the following steps:
 - a. Click the modifier name in the **Tree** view.
 - b. In the left tool bar, click **New Reference to Element**  or **New Reference to Group** . Stylus Studio displays a pop-up menu that lists the elements or groups defined in the schema.
 - c. Double-click the element or group you want to reference.
8. To define an attribute in this complexType:
 - a. Click the name of the complexType in the **Tree** view.
 - b. In the left tool bar, click **New Attribute** . In the **Tree** view, Stylus Studio displays a field for the new attribute.
 - c. Type a name for the new attribute and press Enter. Stylus Studio displays a pop-up menu that lists the built-in simpleTypes and the simpleTypes you defined in this schema.
 - d. Double-click the type of the new attribute.
9. To reference an attribute or attributeGroup in this complexType:
 - a. Click the name of the complexType in the **Tree** view.
 - b. In the left tool bar, click **New Reference to Attribute**  or **New Reference to Attribute Group** . Stylus Studio displays a pop-up menu that lists the attributes or attributeGroups defined in the schema.
 - c. Double-click the attribute or attributeGroup you want to reference.

Defining complexTypes That Mix Data and Elements

Suppose you want to define a complexType that mixes elements and data. For example, you have an XML document with contents such as the following:

```
<letter>
  <salutation>
    Dear Mr.
    <name>Robert Smith</name>
  ,
</salutation>
Your order of
  <quantity>1 </quantity>
  <productName>Baby Monitor </productName>
shipped from our warehouse on
  <shipDate>2001-04-21</shipDate>
.
</letter>
```

The `letter` element and `salutation` element have element and data children. You must define complexTypes for both the `letter` and the `salutation` elements. Their **Mixed** property value must be set to `true`. The **Mixed** property is the one that allows an element to contain both elements (`<shipDate>`, for example) and raw data (Dear Mr. for example) as children.

This section describes how to achieve this using both the **Diagram** and **Tree** views.




Diagram View

◆ **To define a complexType that mixes data and elements:**

1. Create a complexType as described in “[Defining complexTypes That Contain Elements and Attributes – Diagram View](#)” on page 547.
2. In the **Properties** window, click the **Mixed** field.
3. In the drop-down menu that appears, click **true**.

Tree View

◆ **In the Tree view, to define a complexType that mixes data and elements:**


1. Click the **Schema** node.
2. In the left tool bar, click **New complexType** . In the **Tree** view, Stylus Studio displays a field for the new complexType.
3. Type a name for this new complexType and press Enter.
4. In the **Properties** window, double-click the **Mixed** field.
5. Double-click **true**.
6. In the left tool bar, click **New Model Group** . Stylus Studio displays a pop-up menu that lists the group modifiers.
7. Double-click the modifier you want. For a description of the modifiers, see “[Model Group Properties in XML Schemas](#)” on page 594. For the rest of this procedure, assume that you double-click the **sequence** modifier. By default, the elements that are children of this node each appear exactly once. If you want an element to be optional, or if you want an element to appear more than once, specify appropriate values for the properties for minimum occurrence and maximum occurrence.
8. For each element that you want this complexType to contain:
 - a. In the left tool bar, click **New Element Definition** . In the **Tree** view, Stylus Studio displays a field for the new element definition.
 - b. Type a name for the new element and press Enter. Stylus Studio displays a pop-up menu that lists the built-in simpleTypes and any types already defined in the schema.
 - c. Double-click the type for the new element.


Defining complexTypes That Contain Only Attributes

An XML Schema allows you to create groups of attributes. This makes it easy to create a complexType that contains only attributes. The first step is to create an attributeGroup. You can then create a complexType and add a reference to the attributeGroup to the complexType.

Diagram View

◆ **To define a complexType that contains only attributes:**

1. Right-click the schema node .
2. In the shortcut menu, select **Add > AttributeGroup**.

Alternatives: This operation is also available from the **XMLSchema > Diagram > Add > AttributeGroup** menu and from the **Add** button .

The new attributeGroup is added to the XML Schema. It is displayed in the diagram and in the text pane (if you have it open). The properties for the new attributeGroup are displayed in the **Properties** window.

3. Right-click the new attributeGroup.
4. In the shortcut menu that appears, select **Add > Attribute**.
The new attribute is added to the attributeGroup.

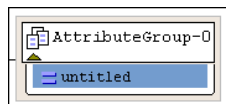






Figure 264. attributeGroup with New Attribute

5. Make sure the new attribute is selected. In the **Properties** window, click the **Data Type** field.
6. Enter or select the type of the attribute.
7. Repeat [Step 3](#) through [Step 6](#) for each attribute that you want to be in the group.
8. Create a complexType as described in “[Defining complexTypes That Contain Elements and Attributes – Diagram View](#)” on page 547.
9. Drag the attributeGroup to the complexType.

Tree View

◆ **To define a complexType that contains only attributes:**

1. Click the **Schema** node.
2. In the left tool bar, click **New Attribute Group** . In the **Tree** view, Stylus Studio displays a field for new attributeGroup.
3. Enter a name for the attributeGroup.
4. In the left tool bar, click **New Attribute Definition** . In the **Tree** view, Stylus Studio displays a field for the new attribute definition.
5. Enter a name for the new attribute. Stylus Studio displays a scrollable, pop-up menu that lists the built-in simpleTypes and any previously defined simpleTypes.
6. Double-click the type of the new attribute.
7. For each additional attribute you want to add to the group, click the name of the attributeGroup in the **Tree** view, and repeat [Step 4](#) through [Step 6](#).
8. Click the Schema node.
9. In the left tool bar, click **New complexType** . In the **Tree** view, Stylus Studio displays a field for the new complexType.
10. Type a name for the new complexType and press Enter.
11. In the left tool bar, click **New Reference to Attribute Group** . Stylus Studio displays a pop-up menu that contains a list of attributeGroups.
12. Double-click the attributeGroup that you want this complexType to contain.

Defining Elements and Attributes in XML Schemas

You can define an element or attribute as part of a complexType. You can also define an element or an attribute as a top-level item. In other words, in the XML document that defines the XML Schema, the element or attribute is a child of the `xsd:schema` element. An element or attribute that is an immediate child of the `xsd:schema` element is a global element or attribute.

A global element or attribute cannot

- Contain a reference to another element or attribute
- Specify values for the `minOccurs`, `maxOccurs`, or use properties

This section covers the following topics:

- [Defining Elements That Carry Attributes and Contain Data in XML Schemas](#) on page 557
- [Defining Elements That Contain Subelements in XML Schemas](#) on page 560
- [Adding an Identity Constraint to an Element](#) on page 561

Defining Elements That Carry Attributes and Contain Data in XML Schemas

You might want to define an element that carries attributes and contains data, but does not contain subelements. In the `purchaseOrder.xsd` document, an example of this is the `internationalPrice` element, shown here in the **Diagram** tab.

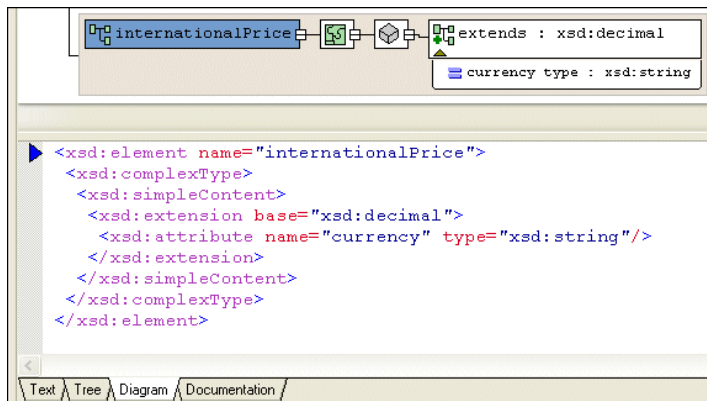




Figure 265. internationalPrice Element in purchaseOrder.xsd

This element has a `currency` attribute, and it contains data based on the `xsd:decimal` simpleType.


Diagram View

◆ To define a complexType that contains only attributes:

1. Right-click the schema node .
2. In the shortcut menu, select **Add > Element**.

Alternatives: This operation is also available from the **XMLSchema > Diagram > Add > Element** menu and from the **Add** button .

The new element is added to the XML Schema. It is displayed in the diagram and in the text pane (if you have it open). The properties for the new element are displayed in the **Properties** window.

3. Create a complexType of the element – right-click the element and select **Add > ComplexType**.
4. Make sure the new complexType is selected.
5. Click the **QuickEdit** button  and select **Derive by extension** or **Derive by restriction**. These choices let you extend or restrict a base simpleType.

The **Type Derivation** dialog box appears.

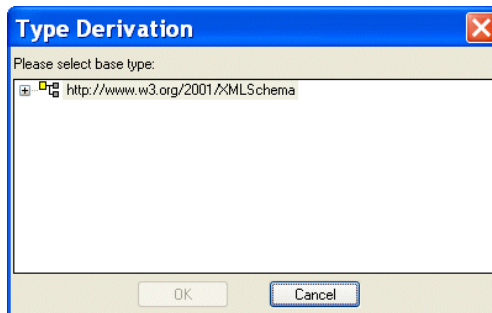



Figure 266. Type Derivation Dialog Box

6. Expand the W3C XML Schema and select the simpleType on which you want to base the data allowed by the complexType.
7. Click OK.

The XML Schema is updated with the element's new definition. [Figure 267](#) shows an extension of the decimal simpleType.








Figure 267. complexType with simpleContent Defined


The simpleContent node () specifies that the complexType can contain only data and attributes. It cannot contain subelements.

8. To add an attribute, right-click the element and select **Add > Attribute**.

Tree View

◆ **To define an element that contains raw data and carries attributes:**

1. Click the **Schema** node.
2. In the left tool bar, click **New Element Definition** . In the **Tree** view, Stylus Studio displays a field for the new element definition.
3. Type the name of the element and press Enter twice. If you press Enter once, Stylus Studio displays a pop-up menu that lists the possible types for the new element. You need to define a new type, so you cannot select from this list. If the pop-up menu does appear, press Enter or click outside the menu. You should now have a named element with no type specified.
4. In the left tool bar, click **New complexType** . In the **Tree** view, Stylus Studio displays a field for the new complexType.
5. In the left tool bar, click **New Content** . Stylus Studio displays a drop-down list.
6. In the drop-down list that appears, double-click **simpleContent**. This is the **Content Type** property. When the content type is **simpleContent**, the complexType you are defining can contain data and attributes. It cannot contain subelements.
7. If you want the contained data to be one of the simpleTypes already defined with no restrictions, click **New Extension**  in the left tool bar.
Stylus Studio displays a scrollable, drop-down list of the simpleTypes built in to XML Schema and previously defined in the current schema.
8. If you clicked **New Extension**, double-click the type of the data you want this element to contain. Go to [Step 9](#).
If you clicked **New Restriction**, follow these steps:
 - a. Double-click the simpleType whose values you want to restrict.
 - b. In the left tool bar, click **New Facet** . Stylus Studio displays a pop-up menu.
 - c. Double-click the type of facet you want to specify.
 - d. In the **Properties** window, double-click the **Value** field.
 - e. Enter a value for the new facet.
 - f. To add another facet, click the **restriction** node for the simpleType, and repeat [Step b](#).
9. In the left tool bar, click the **complexType** node that you created in [Step 4](#).

10. In the left tool bar, click **New Attribute Definition** . In the **Tree** view, Stylus Studio displays a field for the new attribute definition.
11. Type a name for the new attribute and press Enter. Stylus Studio displays a scrollable, drop-down list of the possible types for the new attribute.
12. Double-click the attribute type. If you want to, specify a value for the attribute's **Default** or **Fixed Value** property in the **Properties** window.
13. To add additional attributes, repeat [Step 9](#) through [Step 12](#).

Defining Elements That Contain Subelements in XML Schemas


An element that contains subelements is a complexType. Consequently, you can define an element that contains subelements in either of the following ways:

- Define a top-level complexType. That is, it is a child of the `xsd:schema` node. In the complexType definition, define the subelements. Elsewhere in the schema, define an element that uses the complexType you defined.
- Define an element that is a child of the `xsd:schema` node or a **Model Group** node. In the element definition, define a complexType that contains your subelements.

To define a complexType that contains elements, see [“Defining complexTypes That Contain Elements and Attributes – Diagram View”](#) on page 547 or [“Defining complexTypes That Contain Elements and Attributes – Tree View”](#) on page 551.

Diagram View

◆ **To define an element and define subelements in the element definition:**

1. Right-click the schema node .

2. In the shortcut menu, select **Add > Element**.

Alternatives: This operation is also available from the **XMLSchema > Diagram > Add > Element** menu and from the **Add** button .

The new element is added to the XML Schema. It is displayed in the diagram and in the text pane (if you have it open). The properties for the new element are displayed in the **Properties** window.

3. Right-click the new element and click **QuickEdit**. Select one of the following from the **QuickEdit** menu:
 - **Add Elements Sequence**



- **Add Elements Choice**
- **Add Elements All**
- **Add Elements Any**

Stylus Studio updates the element definition to include a complexType with the sequence, choice, all, or any element you selected in the previous step.

4. Add subelements to the element you created in [Step 3](#).

Tree View

◆ **In the Tree view, to define an element and define subelements in the element definition:**

1. Click the **Schema** node or a **Model Group (all, any, choice, sequence)** node.
2. In the left tool bar, click **New Element Definition** .
3. Enter the name for your new element. Stylus Studio displays a pop-up menu that lists the built-in simpleTypes and any simple or complexTypes already defined in your schema.
4. Press Enter again. Rather than using a type that is already defined, you want to define a new complexType in the definition of your element. You do not want to assign a type to your new element.
5. In the left tool bar, click **New complexType** .
6. Enter a name for the new type.

At this point, the procedure for defining a complexType in the definition of an element is the same as defining a complexType as a child of the **Schema** node. See “[Defining complexTypes That Contain Elements and Attributes – Tree View](#)” on page 551.

Adding an Identity Constraint to an Element

XML Schemas provide a feature that is similar to the DTD ID identity constraint. In a DTD, the value of an ID attribute must be unique within an XML document. In XML Schemas, the type of an identity constraint can be unique, key, or keyref. You use XPath expressions to define the scope of the constraint.

You associate an identity constraint with an element.

- A unique identity constraint forces the result of evaluation of an XPath expression to be unique. Stylus Studio evaluates the XPath expression against the element for which

you define the identity constraint. If the element is present, the result must be unique among the children of that element.

- A key identity constraint specifies that the fields that form the expression must be present in all instance documents. For example, if a key is based on date and number attributes, the date and number attributes must always be specified.
- A keyref identity constraint is equivalent to the IDREF attribute in DTDs. It specifies that the contents of a field in the instance document is the value of a key that is defined in another document. For example, a Quote document would have a reference to the RFQ that originated it.

This section covers the following topics:

- [Example of an Identity Constraint](#) on page 562
- [Diagram View](#) on page 563
- [Tree View](#) on page 564

Example of an Identity Constraint

Suppose you define the following element in an XML Schema:

```
<element name="purchaseReport">
  <complexType>
    <sequence>
      <element name="parts">
        <complexType>
          <sequence>
            <element name="part" maxOccurs="unbounded">
              <attribute name="number" type="SKU"/>
              <attribute name="vendor" type="xs:string"/>
              <attribute name="quantity" type="integer"/>
            </element>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>
```


In an XML document that uses this schema, you could define the following elements:

```
<purchaseReport>
  <parts>
    <part number="00-01-02" vendor="IBM" quantity="10"/>
    <part number="01-01-02" vendor="BEAS" quantity="1"/>
    ...
  </parts>
</purchaseReport>
```

If you want to enforce that there is just one part element for each product that has been purchased, you add the following to the previous XML Schema example:

```
... [previous definition]
  </sequence>
</complexType>
<unique name="pNumKey">
  <selector xpath="parts/part"/>
  <field xpath="@number"/>
  <field xpath="@vendor"/>
</unique>
</element>
```

The schema validator starts with an initial context set that contains purchaseReport elements. It runs the XPath expression parts/part to obtain the data set to be checked. In this example, this is the two part elements. The schema validator then gathers the values from the number and vendor attributes, and builds a key from these values. It then uses the key to check that there are no part elements that have the same tuple.

Diagram View




◆ To specify an identity constraint:

1. Right-click the element for which you want to specify the identity constraint.
2. Select **Add >** and then **Key**, **KeyRef**, or **Unique** from the menu.
3. Right-click the new identity constraint, and select **Selector**.
4. In the **Properties** window, specify the XPath expression that identifies the set of elements to which the identity constraint applies.
5. Return to [Step 3](#) and select **Field**.

6. In the **Properties** window, specify the XPath expression that identifies the element or attribute for each element identified by the selector element that has to be unique.

Tree View

◆ **To specify an identity constraint:**

1. Click the element for which you want to specify the identity constraint.
2. In the XML Schema left-side tool bar, click .
3. In the drop-down list that Stylus Studio displays, double-click **unique**, **key**, or **keyref**.
4. In the **Properties** window, double-click the **Name** field and enter a name for the identity constraint.
5. If you selected **keyref**, then in the **Properties** window, double-click the **Refer** field and enter the name of the key definition.
6. In the tree representation, click the identity constraint you just defined.
7. In the left tool bar, click **New Selector/Key** .
8. In the drop-down list that Stylus Studio displays, double-click **selector**. You must define exactly one selector for each identity constraint.
9. In the **Properties** window, double-click the **XPath Expression** field and enter an XPath expression that returns the element for which you are specifying a constraint.
10. Click the **unique**, **key**, or **keyref** identity constraint you defined in Step 3.
11. In the left tool bar, click **New Selector/Key** .
12. In the drop-down list that Stylus Studio displays, double-click **field**. You must define one or more fields for each identity constraint. A field can be whatever the XPath expression (defined in the next step) retrieves.
13. In the **Properties** window, double-click the **XPath Expression** field and enter an XPath expression that returns the element or attribute that is the key or one of the keys for the constraint. XPath expressions associated with fields return the data that define the key for each element returned by the selector XPath expression.
14. Repeat [Step 10](#) through [Step 13](#) for each additional key field.

Defining Groups of Elements and Attributes in XML Schemas

The *XML Schema Recommendation* allows you to specify groups of elements and groups of attributes. Here is an example of an element group, `purchaseType`:

```
<xsd:group name="purchaseType">
  <xsd:choice>
    <xsd:element name="retail"/>
    <xsd:element name="internet"/>
    <xsd:element name="mailOrder"/>
  </xsd:choice>
</xsd:group>
```

Specification of a group makes it easier to update the schema. You only need to update the group definition. There is no need to change the references to the group.

Here is an example of an attributeGroup, `deliveryDetail`.


```
<xsd:attributeGroup name="deliveryDetail">
  <xsd:attribute name="method"/>
  <xsd:attribute name="vendor"/>
  <xsd:attribute name="dateShipped"/>
  <xsd:attribute name="dateArrived"/>
</xsd:attributeGroup>
```


This section discusses the following topics:

- [Defining Groups of Elements in XML Schemas – Diagram View](#) on page 565
- [Defining Groups of Elements in XML Schemas – Tree View](#) on page 566
- [Defining attributeGroups in XML Schemas – Diagram View](#) on page 567
- [Defining attributeGroups in XML Schemas – Tree View](#) on page 568

Defining Groups of Elements in XML Schemas – Diagram View

◆ To define a group of elements:

1. Define the elements that you want to be in the group. See “[Defining Elements and Attributes in XML Schemas](#)” on page 556.
2. Right-click the schema node .
3. In the shortcut menu, select **Add > Group**.

Alternatives: This operation is also available from the **XMLSchema > Diagram > Add > Group** menu and from the **Add** button .

The new group is added to the XML Schema. It is displayed in the diagram and in the text pane (if you have it open). The properties for the new group are displayed in the **Properties** window.



4. Specify the group name in the **Name** property in the **Properties** window.
5. Right-click the new group.
6. In the shortcut menu, select **Add >** and then select the modifier for the group of elements – **All**, **Choice**, or **Sequence**. In the **Properties** window, click the fields for the **Min Occur.** and **Max Occur.** properties to specify their values. These properties determine how often the group of the elements with the selected modifier can appear.
7. Drag the elements defined in [Step 1](#) and drop them on the all, choice, or sequence modifier created in [Step 6](#).


Alternative

If you prefer, you can create the element group first, define the modifier for the group elements, and then add new elements to the group by right-clicking on the modifier and selecting **Add > Element**. If you do this, you must then define each of the elements you added to the group.

Defining Groups of Elements in XML Schemas – Tree View

◆ To define a group of elements:

1. Define the elements that you want to be in the group. See [“Defining Elements and Attributes in XML Schemas”](#) on page 556.
2. Click the **Schema** node.
3. In the left tool bar, click **New Group** . In the **Tree** view, Stylus Studio displays a field for the new group.
4. Type a name for the group of elements and press Enter.
5. In the left tool bar, click **New Model Group** . Stylus Studio displays a pop-up menu that lists the model group modifiers. See [“Model Group Properties in XML Schemas”](#) on page 594.
6. Double-click a modifier that applies to at least one element that will be in the group.


7. In the **Properties** window, double-click the fields for the **Min Occur.** and **Max Occur.** properties to specify their values. These properties determine how often the subgroup of the elements with the selected modifier can appear.
8. For each element that you want to apply the selected modifier to, perform these steps:
 - a. Click **New Reference to Element** . Stylus Studio displays a pop-up menu that lists the elements defined in the schema.
 - b. Double-click the element you want to add to the group.
 - c. Click the modifier to add another element reference.
9. To add more elements to the group and specify a different modifier for them, click the name of the group in the **Tree** view, and repeat [Step 5](#) through [Step 8](#).


In any location where you can add a model group, you can also add a reference to a model group definition.

Defining attributeGroups in XML Schemas – Diagram View

You define attributeGroups in much the same way that you define element groups – by creating the attributes you want to add to the attributeGroup, creating the attributeGroup, and then dragging-and-dropping the attributes in the attributeGroup. As with element groups, you can define the attributeGroup first and then add new attributes to it, if you prefer. The following procedure describes how to create an attributeGroup by creating the attributes at the same time you create the attributeGroup.

◆ To define an attributeGroup:

1. Right-click the schema node .
2. In the shortcut menu, select **Add > AttributeGroup**.

Alternatives: This operation is also available from the **XMLSchema > Diagram > Add > AttributeGroup** menu and from the **Add** button .

The new attributeGroup is added to the XML Schema. It is displayed in the diagram and in the text pane (if you have it open). The properties for the new attributeGroup are displayed in the **Properties** window.

3. Right-click the new attributeGroup.

4. In the shortcut menu that appears, select **Add > Attribute**.
The new attribute is added to the attributeGroup.

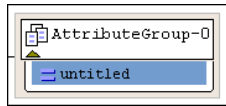




Figure 268. attributeGroup with New Attribute

5. Make sure the new attribute is selected. In the **Properties** window, click the **Data Type** field.
6. Enter or select the type of the attribute.
7. Repeat [Step 3](#) through [Step 6](#) for each attribute that you want to be in the group.

Defining attributeGroups in XML Schemas – Tree View

◆ To define an attributeGroup:

1. Click the **Schema** node.
2. In the left tool bar, click **New Attribute Group** . In the **Tree** view, Stylus Studio displays a field for the new attributeGroup.
3. Type a name for the attributeGroup and press Enter.
4. In the left tool bar, click **New Attribute Definition** . In the **Tree** view, Stylus Studio displays a field for the new attribute definition.
5. Type a name for the new attribute and press Enter. Stylus Studio displays a scrollable, pop-up menu that lists the built-in simpleTypes and any previously defined simpleTypes.
6. Double-click the type of the new attribute.
7. For each additional attribute you want to add to the group, click the name of the attributeGroup in the **Tree** view, and repeat [Step 4](#) through [Step 6](#).

Adding Comments, Annotation, and Documentation Nodes to XML Schemas


The *XML Schema Recommendation* provides comment and annotation nodes for you to provide information that documents an XML Schema. You can add these nodes to any node in an XML Schema.

The difference between comments and annotations is that a human being must read a comment node for it to have meaning. An annotation element allows you to specify nodes that a stylesheet can operate on.

Comments

You cannot add comments in the **Diagram** tab.

◆ **To add comments in the Tree tab:**

1. Click any node in your schema.
2. In the left tool bar, click **New Comment** . In the **Tree** view, Stylus Studio displays a field for the comment.
3. Type your comment and press Enter.

Annotations




You use an annotation element to provide information about the XML Schema. You can annotate any node in your XML Schema. The annotation element always contains at least one appInfo or documentation node. Any text you want to enter must be entered in one of these nodes.

Diagram View

When you create an annotation in the **Diagram** tab, you create the element and specify its subelement (appInfo or documentation) in the **Diagram** tab. You can further describe the node

- In the **Diagram** tab, by editing the node properties directly
- In the **Properties** window
- In the **Text** pane

◆ **To add an annotation:**





1. Right-click the node you want to annotate.
2. Select **Add > Annotation** from the shortcut menu.
The annotation icon appears in the **Diagram** tab .
3. Right-click the annotation icon and select the type of annotation you want to define – appInfo () or documentation () .
4. In the text pane, type the text for the appInfo or documentation node.

Tip

Stylus Studio's backmapper identifies the line representing the element you created in [Step 3](#) in the text pane on the **Diagram**.

Tree View

◆ **To add an annotation:**

1. Click the node you want to annotate.
2. In the left tool bar, click **New Annotation** . Stylus Studio creates and selects an Annotation node.
3. In the left tool bar, click **New Documentation**  or **New Application Info** .
4. If you added documentation, in the **Properties** window, double-click the **Source** field and type the URL or file path for the documentation you want to include in the schema, and press Enter.
Double-click the **Language** field and enter the language of the contents of the source file.
5. In the left tool bar, click **New Text** . In the **Tree** view, Stylus Studio displays a field for the new text.

Moving a Comment or Annotation

If the parent of the new comment or annotation node has more than one child, you can move the comment or annotation with the up or down arrow. However, you cannot move the comment or annotation out of the scope of its parent.

Example

In an XML Schema, you might have a comment node such as the following:

```
<xsd:schema ...>
  <!-- The following element is .... -->
  <xsd:element name="..."/>
```

The contents of a comment node have meaning only when a person reads them. However, the contents of annotation nodes can be operated on. For example:

```
<xsd:schema ... >
  <xsd:element name="foo">
    <xsd:annotation>
      <xsd:documentation language="en">
        This is a <b>foo</b> element. Use it for ...
      </xsd:documentation>

      <xsd:documentation language="jp">
        xksnjgfyre fvhfdbvhjds
      </xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  ...
```

You can apply an XSLT stylesheet to this XML Schema document. The stylesheet could generate an HTML manual by extracting the Documentation nodes in the desired language:

```
<xsl:stylesheet ... >
  <xsl:template match="xsd:element">
    <xsl:apply-templates select=
      "xsd:annotation/xsd:documentation[@language='en']"/>
  </xsl:template>
  ...
```


Defining Notations


A notation is an unparsed entity. It is a name for something that you cannot express in terms of XML. For example, suppose you have an XML file that represents a press release. You can define a notation named `logo` that points to a JPEG image. You can place the notation in the XML file in the location where you want the logo. See <http://www.w3.org/TR/REC-xml#Notations> for more information on the notation element.

Note A notation element is always described as a child of the schema element.

Diagram View

◆ **To define a notation:**

1. Right-click the schema node .
2. In the shortcut menu, select **Add > Notation**.


Alternatives: This operation is also available from the **XMLSchema > Diagram > Add > AttributeGroup** menu and from the **Add** button .

The new notation is added to the XML Schema. It is displayed in the diagram and in the text pane (if you have it open). The properties for the new notation are displayed in the **Properties** window.

3. Specify the details of the notation node in the **Properties** window.

Tree View

◆ **To define a notation:**

1. Click the schema node.
2. In the left tool bar, click **New Notation** .
3. In the field that Stylus Studio displays, enter the name of the notation.
4. In the **Properties** window, double-click the **Public ID** field and enter the public ID. The public ID is a unique string that refers to the location of the external data, but it leaves the resolution of the location to some interpretations, for example, `MyCompany//LOGO//JPEG`.
5. In the **Properties** window, double-click the **System ID** field and enter the system ID. The system ID is the URL that Stylus Studio uses to physically locate the external data, for example, `http://www.mycompany.com/mylogo.jpg`.

Referencing External XML Schemas



Support for referencing external XML Schemas is available only in Stylus Studio XML Enterprise Suite and Stylus Studio XML Professional Suite.

If you want, you can reference definitions from other XML Schemas in your XML Schema document. You might want to do this if you want to simply reuse existing definitions as-is, or if you want to use an existing definition as the base for a type that you want to modify in your XML Schema.

This section covers the following topics:

- [“Ways to Reference XML Schemas”](#) on page 573
- [“Where You Can Reference XML Schemas”](#) on page 574
- [“Referencing XML Schemas in the Tree View”](#) on page 577
- [“Referencing XML Schemas in the Tree View”](#) on page 577
- [“Redefining Nodes”](#) on page 578

Ways to Reference XML Schemas

There are three ways to reference XML Schema:

- Including
- Importing
- Redefining

This section describes each of these techniques and how they can be used. In these descriptions, we use the term *referenced XML Schema* to indicate the XML Schema that is being included, imported, or redefined; and *base XML Schema* to indicate the XML Schema in which the referenced schema is being included, imported, or redefined.

Including an XML Schema

When reference an XML Schema by *including* it, the included XML Schema augments the base XML Schema. Both documents are effectively combined, and they both define the same XML Schema. complexTypes defined in the included XML Schema can be used as the base for new types – you might use a `periodicals` complexType from the included XML Schema to define `weekly` and `quarterly` types for example. Both the included XML Schema and the base XML Schema must have the same target namespace. You can include multiple XML Schemas in a base XML Schema.

Importing an XML Schema

When you reference an XML Schema by *importing* it, the base XML Schema and the imported XML Schema must have different namespaces. The base XML Schema can reference parts of the imported XML schema using a prefix whose namespace is defined in the imported XML Schema, for example. You can import multiple XML Schemas in a base XML Schema.

Redefining an XML Schema

Referencing an XML Schema by *redefining* it is similar to including it, with one important difference: when you redefine an XML Schema in the base XML Schema, you can redefine the definitions of the referenced XML Schema's complexTypes, simpleTypes, groups, and attributeGroups. For example, suppose you release version 1 of an XML Schema. When you need to release version 2 of the XML Schema, you can reference version 1 by redefining it in version 2, which allows you to change the definition of a given node to include a new attribute.

The original complexTypes, simpleTypes, groups, and attributeGroups in the redefined XML Schema are completely masked. They are redefined using *extensions* and *restrictions*. An extension extends the base type – declaring a new element, for example. A restriction constrains the base type.

Where You Can Reference XML Schemas

You can reference external XML Schemas in the **Tree** or **Diagram** tabs. In the text pane of the **Diagram** tab and the **Tree** tab, Stylus Studio displays the referenced XML Schema, but they do not display its contents. For example, in the **Tree** view, you cannot expand the node for an included XML Schema.

In the **Diagram** tab, Stylus Studio displays complete information for any definitions in referenced XML Schema. You can toggle between diagram views of the base and referenced XML Schemas using the definition browser.

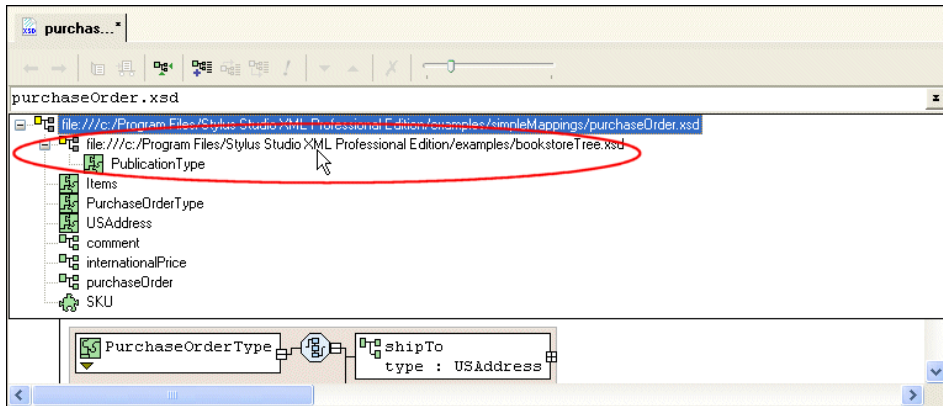



Figure 269. You Can View Referenced Schemas

If you select a referenced schema from the definition browser, as shown in [Figure 269](#), Stylus Studio changes the diagram view to display the referenced schema's structure. If you selected a redefined XML Schema, for example, you could modify its complexType definitions there.

What to Do Next

If you redefine an XML Schema (as opposed to including or importing one), you can redefine nodes after referencing the XML Schema. See [“Redefining Nodes”](#) on page 578 for more information.

Referencing XML Schemas in the Diagram View

- ◆ **To reference an XML Schema in the Diagram view:**
 1. Right-click the schema node .
 2. Click **Referenced Schemas** on the shortcut menu.

The **Referenced Schemas** dialog box appears.

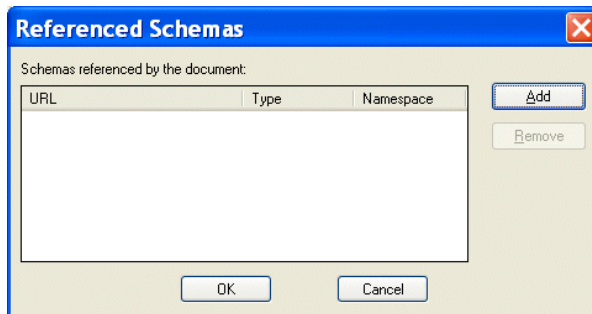


Figure 270. Referenced Schemas Dialog Box

3. Click the **Add** button.

The **Add References to Schema** dialog box appears.

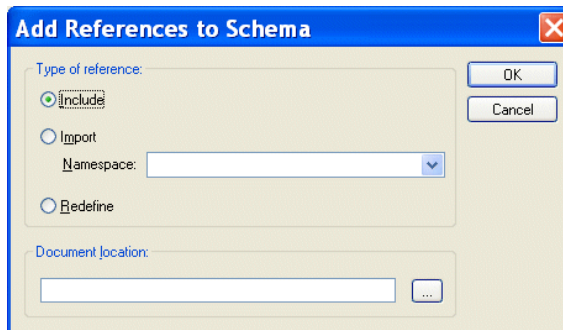


Figure 271. Add References to Schema Dialog Box




4. Specify the type of reference you want to make. If you are redefining an XML Schema, specify the namespace in the **Namespace** field.

5. Specify the URL of the XML Schema you want to reference. For example:
 - myfile.xsd
 - http://www.mycompany.com/schemas/myfile.xsd
 - \\fileservers\schemas\myfile.xsd
6. Click **OK**.

You are returned to the **Referenced Schemas** dialog box.
7. Click **OK** to add the referenced XML Schema to your XML Schema.

Referencing XML Schemas in the Tree View

◆ To reference an XML Schema in the Tree view:

1. Click the **Schema** node.
2. In the XML Schema left-side tool bar, click one of the following:
 - **New Include** 
 - **New Import** 
 - **New Redefine** 
3. In the field that Stylus Studio displays, enter the location. This is a URL that identifies the location of the file that contains the XML Schema. For example, it can be like any one of the following:
 - myfile.xsd
 - http://www.mycompany.com/schemas/myfile.xsd
 - \\fileservers\schemas\myfile.xsd
4. If you defined an **Import** node, in the **Properties** window, double-click the **Target Namespace** field and enter the target namespace. The target namespace must be different from the target namespace of the importing file.

Redefining Nodes

Once you reference an XML Schema by redefining it, you are able to redefine that XML Schema's complexTypes, simpleTypes, groups, and attributeGroups. This section describes how to redefine nodes using the **Diagram** tab.

Extensions and Restrictions

There are two ways to redefine a node: by extension and restriction. An extension extends the base type – adding an element or an attribute definition, for example. A restriction constrains the base type – limiting a type to a certain range of values, for example.

Specifying Restriction Facets

If you define a restriction using a simpleType, the **Properties** window displays a section that allows you to define the facets that restrict that type, as shown in [Figure 272](#).

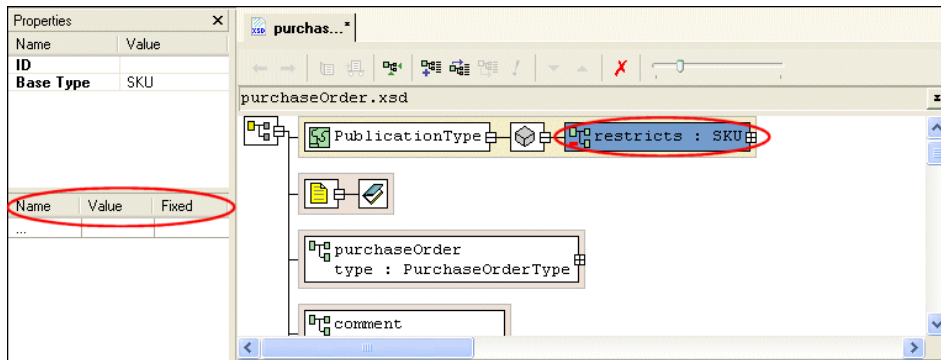



Figure 272. Facets for Describing Restrictions

Restriction facets include length, minLength, maxLength, and totalDigits. For each facet you specify, you provide the facet name, a value, and, for some facets, whether or not the value is fixed. Note that not all facets apply to all types.

See [“About Facet Types for simpleTypes”](#) on page 544 for more information on facets.

How to Redefine a Node

◆ To redefine a node in the Diagram view:

1. Right-click the schema node .
2. Select **Redefine** from the shortcut menu.

The **Redefine Schema Symbols** dialog box appears.

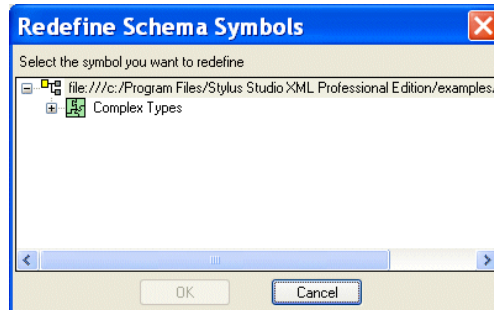


Figure 273. Redefine Schema Symbols Dialog Box

3. Select the node from the redefined XML Schema you want to redefine and click **OK**.
The redefined node is added to the diagram, and the text for the redefined node appears in the text pane. For example, `<xsd:complexType name="PublicationType"/>`.
4. Right-click the redefined node.
5. From the shortcut menu, select **Quick Edit >** and then either **Derive by extension** or **Derive by restriction**.

The **Type Derivation** dialog box appears.

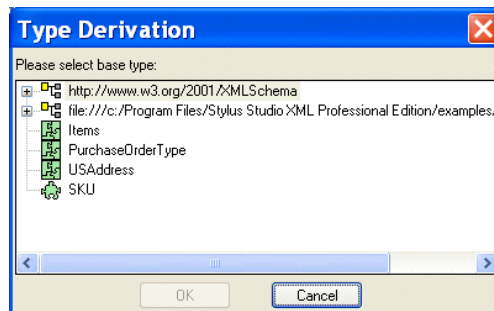


Figure 274. Type Derivation Dialog Box

6. Select the base type from which you want to derive the definition of the node you are redefining, and click **OK**.

The node you added in [Figure 3](#) is modified in the diagram to display the restriction or extension you are using to redefine it, as shown in [Figure 275](#).

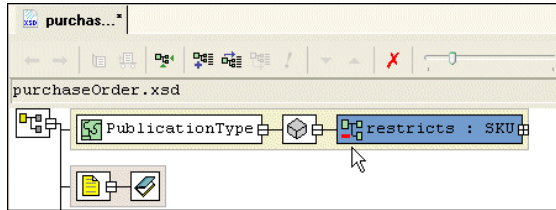


Figure 275. Redefined Node as a Restricted simpleType

The code displayed in the text pane is also modified. For example:

```
<xsd:complexType name="PublicationType">
  <xsd:simpleContent>
    <xsd:restriction base="SKU"/>
  </xsd:simpleContent>
</xsd:complexType>
```

7. If you specified a restriction of a simpleType, specify the restriction facets in the **Properties** window. See [“Specifying Restriction Facets”](#) on page 578 if you need help with this step.

Generating Documentation for XML Schema



The **Documentation** tab is available only in Stylus Studio XML Enterprise Suite and Stylus Studio XML Professional Suite.

The **Documentation** tab of the XML Schema Editor displays HTML documentation that describes the currently active XML Schema. The HTML is presented using the XS3P stylesheet.

- ◆ **To display XML Schema documentation, open an XML Schema document and click the Documentation tab.**

This section covers the following topics:

- [XS3P Stylesheet Overview](#) on page 581
- [Saving XML Schema Documentation](#) on page 584

- [Printing XML Schema Documentation](#) on page 584

XS3P Stylesheet Overview

By default, Stylus Studio displays XML Schema on the **Documentation** tab using the XS3P stylesheet from the DSTC Project Titanium (<http://titanium.dstc.edu.au/>). [Figure 276](#) shows how the purchaseOrder.xsd looks when displayed using this stylesheet.

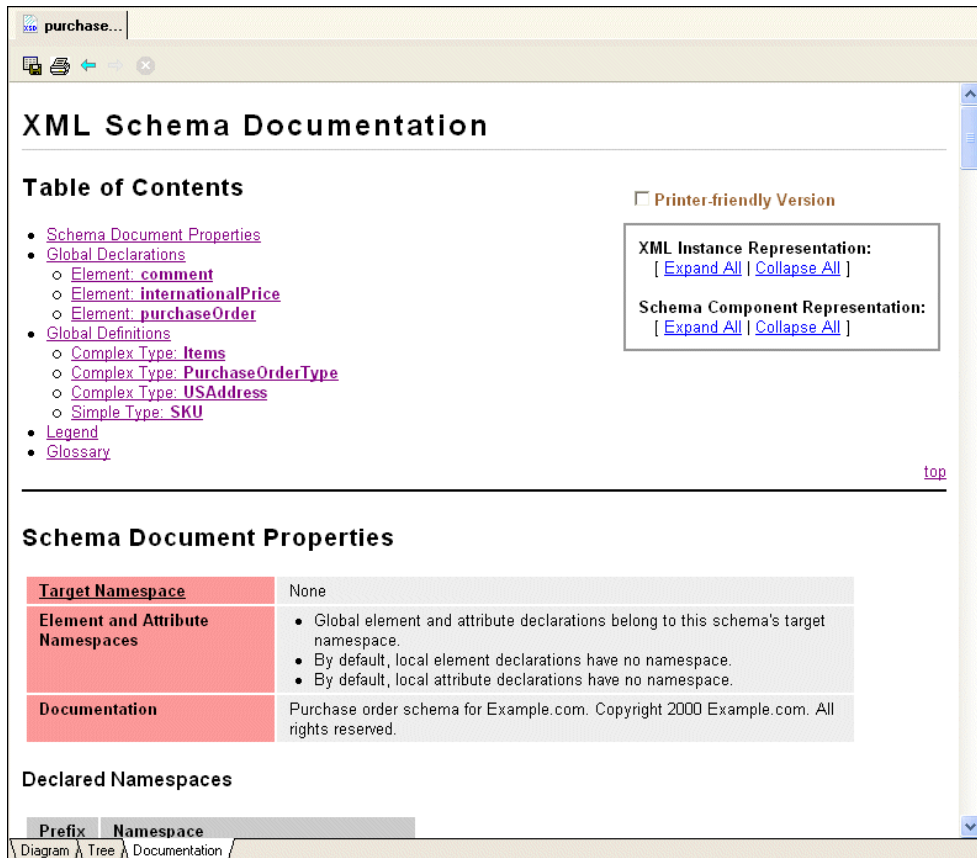


Figure 276. XML Schema Documentation Displayed Using XS3P Stylesheet

The XS3P stylesheet contains

- A customizable title (the default title is *XML Schema Documentation*)
- A table of contents with hypertext links to sections in the documentation

- Information about the XML Schema's properties, such as its target namespace and any declared namespaces
- Global declarations and global definitions, if any
- A *legend* that describes the graphical conventions used in the XML Schema documentation. The legend uses a fictitious type declaration for example purposes.
- A *glossary* that defines terminology used in the XML Schema documentation.

Tip You can hide the legend and the glossary by clicking the **Printer-friendly Version** check box at the top of the page.

This section covers the following topics:

- [XS3P Stylesheet Features](#) on page 582
- [XS3P Stylesheet Settings](#) on page 583
- [Modifying the XS3P Stylesheet](#) on page 584

XS3P Stylesheet Features

The XS3P stylesheet has several features that affect content and layout of XML Schema displayed on the **Documentation** tab. You can

- Create a printer-friendly version of the documentation by clicking the **Printer-friendly Version** check box.

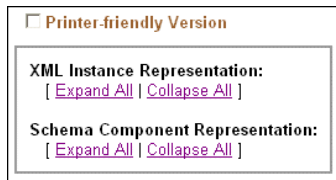


Figure 277. Features of XML Schema Documentation

When you click this check box, Stylus Studio

- Hides the Legend and Glossary sections
- Automatically expands all XML instance and schema component representations
- Removes the expand/collapse controls from the page
- Expand and collapse XML instance and schema component representations. You can do this for every XML instance or schema component by clicking the **Expand All** and

Collapse All buttons associated with these representations. You can also set this option for individual instances by clicking the **+/-** button, as shown in [Figure 278](#).

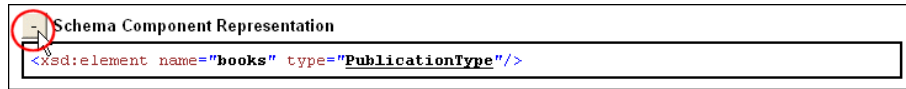


Figure 278. Representations Can be Collapsed/Expanded Individually

- Customize and modify the XML Schema documentation. For example, you can choose to include all super-types, you can change the default name, and you can specify the sort order. Settings for these and other properties that affect the content and appearance of the XML Schema documentation are displayed in the **Options** dialog box. See [XS3P Stylesheet Settings](#) on page 583 for more information.

XS3P Stylesheet Settings

The XS3P stylesheet allows you to modify the following:

- Title – The default title is *XML Schema Documentation*, but you can change it to whatever you want by editing the **Title** field.
- Sort order – By default, Stylus Studio sorts information in the XML Schema in alphabetical order by type name. If you want to display information in document order, set the **Sort by Component** field to **False**.
- Whether or not you want Stylus Studio to search included and imported XML Schemas.
- If you want to incorporate instructions in the HTML, the **Use JavaScript** option makes it easy for you to add instructions such to display pop-up windows and hide information in the generated HTML document.
- Inclusion in the XML Schema documentation of
 - Supertypes
 - Subtypes
 - Glossary
 - Legend
 - xsd namespace prefix
 - Schema diagrams

You control these settings on the **Documentation** page of the **Options** dialog box.

Tip You can also modify the stylesheet directly. See [Modifying the XS3P Stylesheet](#) on page 584.


Modifying the XS3P Stylesheet

You can customize the XS3P stylesheet that Stylus Studio uses to display XML Schema documentation. The XS3P stylesheet is in the \schema-documentation directory where you installed Stylus Studio: bin\Plugins\schema-documentation. The name of the stylesheet file is xs3p.xsl.

Should you choose to modify the default XS3P stylesheet, the new stylesheet must have the same name as the original. After you modify and save this file, click the refresh button on the Stylus Studio tool bar to see your changes.


Tip Make a copy of the xs3p.xsl file before you modify it.

Saving XML Schema Documentation

To save the XML Schema documentation, click the **Save Documentation** button () in the XML Schema window. The XML Schema documentation is saved as an HTML file that you can edit and add to as you would any other HTML file.

Printing XML Schema Documentation

◆ **To print XML Schema documentation:**

1. If you are using the XS3P stylesheet, optionally click the **Printer-friendly Version** check box at the top of the XML Schema documentation.
2. Review the settings on the **Documentation** page of the **Options** dialog box (**Tools > Options > Module Settings > XML Schema Editor > Documentation**).
3. Optionally, preview the XML Schema documentation (**File > Print Preview**).
4. Click the **Print** button () , or type Ctrl + P.

Generating JAXB Classes

You can generate JAXB (Java Architecture for XML Binding) application class files from an XML Schema. The generated application skeleton (`Main.java`, for example) demonstrates how to use `Marshaller` and `Unmarshaller` classes.

For more information on using JAXB, refer to the Sun Microsystems' Java Technology and XML documentation, located here: <http://java.sun.com/xml/jaxb/docs.html>.

What Stylus Studio Generates

By default, Stylus Studio creates the Java package in the directory in which the source XML Schema file (.xsd) resides. The package name is the XML Schema file name, and the default JAXB application class name is Main. Also by default, the generated files are added to the current Stylus Studio project in a folder with the same name as the package.

You can change any of these values at the time you run the JAXB code generator.

How to Generate JAXB Classes

◆ **To generate JAXB classes:**

1. Open the XML Schema file for which you want to generate JAXB classes.
2. Click the **Diagram** tab if it is not already selected.
3. Select **XMLSchema > Generate Java** from the Stylus Studio menu.

The **Generate Java Binding Class** dialog box appears.

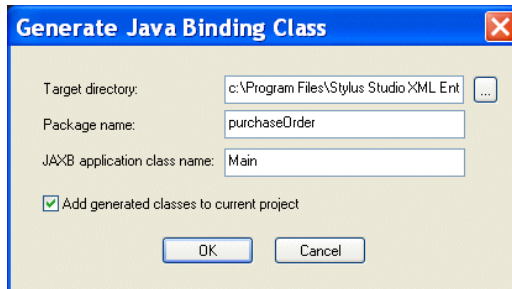


Figure 279. Generate Java Binding Class Dialog Box

4. If necessary, change the default values.
5. Click **OK**.

Stylus Studio generates JAXB classes. The **Output** window displays code generation status.

Compiling JAXB Class Files

The following procedure assumes that you created the JAXB package as part of the current Stylus Studio project.

◆ **To compile JAXB class files:**

1. Display the **Project** window if it is not already open.
2. Right-click the folder that represents the JAXB package.
3. Select **Compile** from the shortcut menu.

Stylus Studio compiles the JAXB package. The **Output** window displays compile status.

About XML Schema Properties



The **Documentation** tab is available only in Stylus Studio XML Enterprise Suite and Stylus Studio XML Professional Suite.

When the **Diagram** or **Tree** tab of an XML Schema is active, you can see the properties for the selected node. Click the node whose properties you want to view, and the properties appear in the **Properties** window. If the **Properties** window is not visible, select **View > Properties** from the menu bar.

To change the value of a property, click the property field and enter the new value. If only certain values are allowed, Stylus Studio displays a drop-down list of the valid choices. In the **Diagram** view, some properties are read-only. To modify these properties, switch to the **Tree** view and change the property there. When you switch back to the **Diagram** view, your change is visible.

Each type of node has its own set of properties. The following topics describe the properties for various node types:

- [About xsd:schema Properties](#) on page 588
- [Element and Element Reference Properties in XML Schemas](#) on page 590
- [Attribute and Attribute Reference Properties in XML Schemas](#) on page 592
- [Group Properties in XML Schemas](#) on page 594
- [Model Group Properties in XML Schemas](#) on page 594
- [Complex and simpleType Properties in XML Schemas](#) on page 596
- [Restriction and Extension Type Properties in XML Schemas](#) on page 597

- [Content Type Properties in XML Schemas](#) on page 597
- [Aggregator Type Properties in XML Schemas](#) on page 598
- [Facet Type Properties in XML Schemas](#) on page 599
- [Notation Type Properties in XML Schemas](#) on page 600
- [Include Type Properties in XML Schemas](#) on page 600
- [Import Type Properties in XML Schemas](#) on page 601
- [Redefine Type Properties in XML Schemas](#) on page 601
- [Identity Constraint Type Properties in XML Schemas](#) on page 601
- [Constraint Element Type Properties in XML Schemas](#) on page 602
- [Documentation Type Properties in XML Schemas](#) on page 602

About `xsd:schema` Properties

The root element of every XML Schema document is the `xsd:schema` element. The `xsd:schema` element has the properties described in [Table 61](#). Click the **Tree** tab, and then click the **Schema** node to view the properties for the `xsd:schema` element.

Table 61. `xsd:schema` Properties

<i>Property</i>	<i>Description</i>
Type	The type is always Schema.
Namespace	The namespace for the Schema node is usually <code>xsd</code> , but you can change it.
Target Namespace	This is the namespace that elements and attributes defined in an instance document belong to. For example, suppose you define the following: <pre><xsd:schema ... targetNamespace="http://myNS"> <xsd:element name="myelement"/></pre> In an instance document, the following declarations conform with the target namespace: <pre><myelement xmlns="http://myNS"/> <myns:myelement xmlns:myns="http://myNS"/></pre> However, the following declaration does not conform: <pre><myns:myelement xmlns:myns="http://anotherNS"/></pre>
Version	Use this property as a convenient way to track the revisions of your XML Schema document.

Table 61. xsd:schema Properties

<i>Property</i>	<i>Description</i>
Default Element Form	<p>An element or attribute's form is either <i>qualified</i> or <i>unqualified</i>. A form of <i>qualified</i> means that each time an element or attribute is referenced in the schema document, you must specify the prefix of its namespace. Every element and attribute has a <i>form</i> attribute. If it is not explicitly defined, the schema processor checks the default attribute form specified for the Schema node. For example:</p> <pre><xs:schema elementFormDefault="qualified" targetNamespace="http://myNs" xmlns:myns="http://myNS"> <xs:element name="topElem"> </xs:element> <xs:element name="anElem"> <xs:complexType> <xs:sequence> <xs:element ref="myns:topElem"> . . .</pre> <p>If the form for the element <code>topElem</code> (or, the default form for elements) was defined to be <i>unqualified</i>, the reference could have used <code>ref="topElem"</code>.</p>
Default Attribute Form	
Default Blocked Definitions	
Default Final Definitions	If an element does not have its own <i>blocked</i> or <i>final</i> definition, the schema processor uses the default <i>blocked</i> or <i>final</i> definition you specify here.

Element and Element Reference Properties in XML Schemas

Both global and local elements have the properties described in [Table 62](#). References to elements have the same properties except where noted.

Table 62. Element and Element Reference Properties

<i>Property</i>	<i>Description</i>
Type	For elements, the type is always <code>Element</code> . For references to elements, the type is always <code>Ref. to Element</code> .
Name	The tag name you use in an instance document. Specify the name you want the element to have.
Min Occur.	Specifies the minimum number of instances of this element that can be present. If an element is not required to be present, specify 0. You cannot specify this property for a global element. If you do, Stylus Studio ignores it.
Max Occur.	Specifies the maximum number of instances of this element that can be present. If there is no limit to the number of instances, specify unbounded. You cannot specify this property for a global element. If you do, Stylus Studio ignores it.
Data Type	The type of the data that the element contains. Select from all <code>simpleTypes</code> defined in an XML Schema, and all types (simple or complex) that you define in the same schema. Nodes that are references to elements do not have this property.
Default	Specifies the default value for this element. Specification of this property makes sense only for optional elements. If you specified 0 for the <code>Min Occur.</code> property, you can specify a default value. When this element is in an instance document, the element has whatever value you specify. If you do not specify this element, the schema processor behaves as though you had specified it with the default value. When you specify a default value for an element, that element must be optional in an instance document. An element can have a value for the <code>Default</code> property or a value for the <code>Fixed Value</code> property. The two properties are mutually exclusive.

Table 62. Element and Element Reference Properties

<i>Property</i>	<i>Description</i>
Fixed Value	When you specify a value for Fixed Value, it is optional for the element to appear in an instance document. However, if the element does appear, it must have the value specified by Fixed Value. Whether or not you specify this element in an instance document, the schema processor behaves as though you had specified this element with the fixed value. An element can have a value for the Fixed Value property or a value for the Default property. The two properties are mutually exclusive.
Abstract	A Boolean value that indicates whether substitution for this element is required. When Abstract is true, the element cannot be used in an instance document. Instead, a member of the element's substitution group must appear in the instance document.
Nilable	A Boolean value that indicates whether the contents of the element can be set to nil. A value of true indicates that the element can be empty; that is, it is permissible for the element to not contain any subelements, attributes, or data.
Form	An element's form is either qualified or unqualified. A form of qualified means that each time the element is referenced in the schema document, you must specify the prefix of its namespace. Every element has a form attribute. If it is not explicitly defined, the schema processor checks the default attribute form specified for the Schema node.
Blocked Substitutions	Defines that this element cannot be derived in some forms. That is, it specifies that one or more extensions, restrictions or substitutions cannot be permitted. For example, an enumeration for all the states in the United States can block extensions and substitutions, thus allowing derived data types only so as to restrict the number of valid states.

Table 62. Element and Element Reference Properties

<i>Property</i>	<i>Description</i>
Final Substitutions	Specifies that this element is not allowed to be substituted in a substitution group if these are extensions or restrictions of the same base type. For example, suppose an Invoice contains a reference to a PO document. The PO document is derived from AccountingDocument. If PO document has the final="extensions" attribute, and PartialPO is defined as an extension from AccountingDocument, the Invoice cannot substitute PO with PartialPO.
Substitution Groups	If an element defines an element name definition in a substitution group, it means that it can be used in all the places where there is a reference to that element. For example, suppose the PO document definition indicates that it can refer to an RFQ element. You can specify that a foo element is in the substitution group for an RFQ element. If you do, a PO document is valid if it refers to a foo element.

Attribute and Attribute Reference Properties in XML Schemas

Both global and local attributes have the properties described in the [Table 63](#). References to attributes have the same properties, except where noted.

Table 63. Attribute and Attribute Reference Properties

<i>Property</i>	<i>Description</i>
Type	For attributes, the type is always Attribute. For attribute references, the type is always Ref. to Attribute.
Name	The attribute name you use in an instance document. Specify the name you want the attribute to have.
Data Type	The type of the data that is the value of the attribute. Select from all simpleTypes defined in an XML Schema, and all simpleTypes that you already defined in the same schema. References to attributes do not have this property.

Table 63. Attribute and Attribute Reference Properties

<i>Property</i>	<i>Description</i>
Default	<p>Specifies the default value for this attribute. Specification of this property makes sense only for optional attributes. If you specified optional for the Restrictions property, you can specify a default value.</p> <p>If this attribute is in an instance document, the attribute has whatever value you specify. If you do not specify this attribute, the schema processor behaves as though you had specified it with the default value. When you specify a default value for an attribute, that attribute must be optional in an instance document. An attribute can have a value for the Default property or a value for the Fixed Value property. The two properties are mutually exclusive.</p>
Fixed Value	<p>When you specify a value for Fixed Value, it is optional for the attribute to appear in an instance document. However, if the attribute does appear, it must have the value specified by Fixed Value. Whether or not you specify this attribute in an instance document, the schema processor behaves as though you had specified this attribute with the fixed value. An attribute can have a value for the Fixed Value property or a value for the Default property. The two properties are mutually exclusive.</p>
Restrictions	Specify prohibited, optional, or required.
Form	<p>An attribute's form is either qualified or unqualified. A form of qualified means that each time the attribute is referenced in a schema document, you must specify the prefix of its namespace. Every attribute has a form attribute. If it is not explicitly defined, the schema processor checks the default attribute form specified for the Schema node.</p>

Group Properties in XML Schemas

A group contains references to elements. Groups have the properties described in [Table 64](#):

Table 64. Group Properties

<i>Property</i>	<i>Description</i>
Type	The type is always Group.
Name	The name you specified for the group.
Min Occur .	Specifies the minimum number of instances of this group that can appear in a complexType that references this group. If a group is not required to be present, specify 0.
Max Occur .	Specifies the maximum number of instances of this group that can appear in a complexType that references this group. If there is no limit to the number of instances, specify unbounded.

Model Group Properties in XML Schemas

After you create a group node or a complexType node, you can add a model group node as a child. A model group specifies rules for the occurrence of elements. These are the elements that are the children of the group or complexType in an instance document. A

model group references and defines elements. Model groups have the properties described in [Table 65](#):

Table 65. Model Group Properties

<i>Property</i>	<i>Description</i>
Type	The type is always Model Group.
Modifier	<p>Specifies the occurrence rules for the elements that you add as children of the model group node. Specify one of the following values:</p> <ul style="list-style-type: none"> ● all specifies that each element must appear exactly zero or one time. The elements can appear in any order. In an instance document, the children of the group or complexType can include 0 or 1 instance of each element. ● choice specifies that exactly one element can be present, and there must be only one instance of that element. In an instance document, exactly one element can be a child of the group or complexType. ● sequence specifies that the elements must appear in the order in which they are specified in the schema. For each element, you can specify whether it is optional and whether it can appear more than once. The default is that exactly one must be present in an instance document. In an instance document, each element that appears must be in the same order as in the schema. <p>Another value you can specify is any. When you specify any, you do not add any element definitions or references to elements. As the name implies, any element can appear any number of times.</p>
Min Occur.	Specifies the minimum number of instances of this model group that can appear in this group or complexType. If a model group is not required to be present, specify 0.
Max Occur.	Specifies the maximum number of instances of this model group that can appear in this group or complexType. If there is no limit to the number of instances, specify unbounded.

Complex and simpleType Properties in XML Schemas

complexTypees have the properties described in [Table 66](#). simpleTypes have only the Type and Name properties.

Table 66. Complex and simpleType Properties

<i>Property</i>	<i>Description</i>
Type	The type is always complexType or simpleType.
Name	The type name you use elsewhere in the XML Schema. Specify the name you want the type to have.
Abstract	A Boolean value that indicates whether substitution for this complexType is required. When Abstract is true, the complexType cannot be used in an instance document. Instead, a member of the complexType's substitution group must appear in the instance document. simpleTypes do not have this property.
Mixed	A Boolean value that indicates whether or not this complexType can contain raw data as well as elements and attributes. A value of true indicates that it can contain raw data. simpleTypes do not have this property.
Blocked Substitutions	Defines that this type cannot be derived in some forms. That is, it specifies that one or more extensions, restrictions or substitutions cannot be permitted. For example, an enumeration for all the states in the United States can block extensions and substitutions, thus allowing derived data types only so as to restrict the number of valid states.
Final Substitutions	Specifies that the type is not allowed to be substituted in a substitution group if these are extensions or restrictions of the same base type. For example, suppose an Invoice contains a reference to a PO document. The PO document is derived from AccountingDocument. If PO document has the final="extensions" attribute, and PartialPO is defined as an extension from AccountingDocument, the Invoice cannot substitute PO with PartialPO.

Restriction and Extension Type Properties in XML Schemas

When you define a `simpleType`, you always derive it from a built-in XML Schema `simpleType`, or a `simpleType` you previously defined. To specify the `simpleType` that your `simpleType` is based on, add a restriction node or an extension node to your `simpleType` node.

A restriction node indicates that your `simpleType` is a subset of some other `simpleType`. An extension node indicates that your `simpleType` extends the range of values provided by an existing `simpleType`.

Restriction type nodes and extension type nodes have the properties described in [Table 67](#):

Table 67. Restriction and Extension Type Properties

<i>Property</i>	<i>Description</i>
Type	The type is always <code>Restriction</code> or <code>Extension</code> .
Base Type	Indicates the data type that this <code>simpleType</code> is based on.

Content Type Properties in XML Schemas

Content types have the properties described in [Table 68](#):

Table 68. Content Type Properties

<i>Property</i>	<i>Description</i>
Type	The type is always <code>Content</code> .
Mixed	A Boolean value that indicates whether this node can contain text as well as elements.
Content Type	When the value is <code>simpleContent</code> , the node can contain only character data and no elements or attributes. When the value is <code>complexContent</code> , the node can contain character data, elements, and attributes.

Aggregator Type Properties in XML Schemas

After you create a `simpleType` node, you can add an aggregator node as its child. An aggregator node indicates that a single instance of an element of your new `simpleType` contains a sequence of atomic types. Aggregator types have the properties described in [Table 69](#):

Table 69. Aggregator Type Properties

<i>Property</i>	<i>Description</i>
Type	Must be <code>list</code> or <code>union</code> . <ul style="list-style-type: none">● <code>list</code> indicates that all instances in the sequence must be of the same type.● <code>union</code> indicates that the instances in the sequence can be of different types.
Aggregator Type	The type of the instances included in your new <code>simpleType</code> . If the value of <code>Type</code> is <code>union</code> , you can specify a space-separated list.

Facet Type Properties in XML Schemas

Facet types have the properties described in [Table 70](#):

Table 70. Facet Type Properties

<i>Property</i>	<i>Description</i>
Facet Type	Must be one of the following: enumeration, fractionDigits, length, maxExclusive, maxInclusive, maxLength, minExclusive, minInclusive, minLength, pattern, totalDigits, or whiteSpace.
Fixed	<p>A Boolean value that indicates whether you can further restrict the simpleType with this same facet and a different value. The default is false. That is, the default is that you can apply the same facet more than once. For example, suppose you specify the following definition:</p> <pre><xsd:simpleType name="zip"> <xsd:restriction base="xsd:string"> <xsd:length value="5" fixed="true"/> </xsd:restriction> </xsd:simpleType></pre> <p>This defines a postal code whose length is 5 characters. You can further restrict this simpleType with, for example, the pattern facet so that the first three characters must always be "100", but you cannot further restrict the length facet when the Fixed property is set to true.</p> <p>Facet types of pattern and enumeration do not have the Fixed property.</p>
Value	Varies according to the facet type. See “About Facet Types for simpleTypes” on page 544.

Notation Type Properties in XML Schemas

Notation types have the properties described in [Table 71](#). See “[Defining Notations](#)” on page 572 for more information.

Table 71. Notation Type Properties

<i>Property</i>	<i>Description</i>
Type	The type is always Notation.
Name	The name you specify for the notation.
Public ID	Unique string that refers to the physical location of the external data, for example, MyCompany//LOGO//JPEG.
System ID	URL used to physically locate the external data, for example, http://www.mycompany.com/mylogo.jpg .

Include Type Properties in XML Schemas

Include types have the properties described in [Table 72](#). See “[Referencing External XML Schemas](#)” on page 573 for more information.

Table 72. Include Type Properties

<i>Property</i>	<i>Description</i>
Type	The type is always Include.
Location	URL that identifies the location of the file that contains the XML Schema.

Import Type Properties in XML Schemas

Import types have the properties described in [Table 73](#). See “[Referencing External XML Schemas](#)” on page 573 for more information.

Table 73. Import Type Properties

<i>Property</i>	<i>Description</i>
Type	The type is always Import.
Location	URL that identifies the location of the file that contains the XML Schema.
Target Namespace	This is the namespace that elements and attributes defined in an instance document belong to.

Redefine Type Properties in XML Schemas

Redefine types have the properties described in [Table 74](#). See “[Referencing External XML Schemas](#)” on page 573 for more information.

Table 74. Redefine Type Properties

<i>Property</i>	<i>Description</i>
Type	The type is always Redefine.
Location	URL that identifies the location of the file that contains the XML Schema.

Identity Constraint Type Properties in XML Schemas

Identity Constraint types have the properties described in [Table 75](#). See “[Adding an Identity Constraint to an Element](#)” on page 561 for more information.

Table 75. Identity Constraint Type Properties

<i>Property</i>	<i>Description</i>
Type	The type is always Identity Constraint.
Name	The name you specify for the identity constraint.

Constraint Element Type Properties in XML Schemas

Constraint Element types have the properties described in [Table 76](#). See “[Adding an Identity Constraint to an Element](#)” on page 561 for more information.

Table 76. Constraint Element Type Properties

<i>Property</i>	<i>Description</i>
Type	The type is always Constraint Element.
XPath Expression	An XPath expression that returns the element for which you are defining a constraint.

Documentation Type Properties in XML Schemas

Documentation types have the properties described in the following table:

Table 77. Documentation Type Properties

<i>Property</i>	<i>Description</i>
Type	The type is always Documentation.
Source	A path or URL for an external file that contains the documentation.
Language	The language of the contents of the documentation.

Chapter 8 **Defining Document Type Definitions**

This section provides information about how to use the Stylus Studio Document Type Definition (DTD) editor to define a DTD. Familiarity with DTDs is assumed.

This section discusses the following topics:

- [“What Is a DTD?”](#) on page 604
- [“Creating DTDs”](#) on page 604
- [“About Editing DTDs”](#) on page 605
- [“About Modifiers in Element Definitions in DTDs”](#) on page 606
- [“Defining Elements in DTDs”](#) on page 609
- [“Defining General Entities and Parameter Entities in DTDs”](#) on page 618
- [“Inserting White Space in DTDs”](#) on page 621
- [“Adding Comments to DTDs”](#) on page 621
- [“About Node Properties in DTDs”](#) on page 622
- [“Associating an XML Document with an External DTD”](#) on page 626
- [“Moving an Internal DTD to an External File”](#) on page 626

What Is a DTD?

A document type definition (DTD) describes the structure of a document. It specifies which elements can contain which other elements, which elements are optional and which are required, and which elements contain data. For example, a DTD might specify that a book element

- Must contain exactly one `title` element
- Can contain any number of author elements
- Might contain a `subtitle` element

To use a DTD, you must associate it with an XML document. A DTD can be internal or external. An *internal DTD* is inside the XML document that uses it. It appears in the `DOCTYPE` element, which immediately follows the XML declaration at the beginning of the document. An *external DTD* is in a separate file. An XML document that uses an external DTD specifies the path for the DTD in its `DOCTYPE` element. For example, the following `DOCTYPE` element specifies that `bookstore` is the root element in this XML document, and that the DTD that this document uses is stored in the file system at

C:\mydir\bookstore.dtd:

```
<!DOCTYPE bookstore SYSTEM "file:///C:\mydir\bookstore.dtd">
```

A *document instance* is an XML document that uses a particular DTD. In other words, the contents of a document instance have been tagged according to the structure defined in the DTD it is associated with. For example, if the contents of the `bookstore.xml` file follow the structure defined in the `bookstore.dtd` file, `bookstore.xml` is a document instance of the `bookstore` DTD.

Creating DTDs

To create a DTD, select **File > New > DTD Schema** from the Stylus Studio menu bar. Stylus Studio displays the DTD schema editor.

The Stylus Studio DTD editor provides two views of a DTD. In the **Tree** view, Stylus Studio uses branches and leaves to represent the DTD. When you define a DTD in the **Tree** view, you do not need to know the details about DTD syntax. In the **Text** view, Stylus Studio displays the lines of text that make up the DTD. To define a DTD in the **Text** view, you must be familiar with DTD syntax.

If you are editing an XML document and you want to create a DTD for that document, click the **Schema** tab. Stylus Studio displays the **Schema Not Found** dialog box. Indicate

that you want Stylus Studio to generate a DTD and indicate whether you want the new DTD to be internal or external. After you respond to the prompts and click **Yes**, Stylus Studio automatically creates the DTD for you and displays it in the **Schema** tab.

If you instruct Stylus Studio to create an internal DTD, you can update the DTD in the XML editor. If you instruct Stylus Studio to create an external DTD, you must explicitly open it to update it. An external DTD that Stylus Studio displays in the **Schema** tab is read-only.

To use Stylus Studio to validate an XML document against a DTD, see “[Validating XML Documents](#)” on page 209. If you update a DTD in Stylus Studio and that DTD is associated with an XML document that is open in Stylus Studio, Stylus Studio refreshes the schema information for the XML document.

About Editing DTDs


Stylus Studio displays a DTD with two views. Click the **Text** tab or the **Tree** tab to display the view you want. The **Tree** tab displays a DOM-like tree that represents the DTD.


You can specify and edit the DTD in either view. However, the recommended method is to edit the DTD in the **Tree** view. The **Tree** view provides tools tailored for creating a DTD. The tool bar on the left provides a button for defining each node in a DTD. After you select a node in the tree, the DTD editor allows you to add only those nodes that are valid at that point.

Also, in the **Tree** view of the DTD, you can see the properties for each node. When you click the node whose properties you want to view, the properties appear in the **Properties** window. If the **Properties** window is not visible, select **View > Properties** from the menu bar.

After you add a node, you can make changes in the **Properties** window, the **Tree** tab, or the **Text** tab. Any changes you make in one place are immediately reflected in the other places.

Restrictions

A DTD can include other, external DTDs. In a future release, it is expected that you will be able to click **Open Schema**  to display an included DTD. However, in this release, this is not supported.

DTDs are not XML documents. Consequently, as you would expect, **Indent XML Tags**  does not work on DTDs.

About Modifiers in Element Definitions in DTDs

When you define an element, you specify one or more modifiers. A modifier specifies a rule about the structure or occurrence of the element being defined. An element can have only one top-level modifier. However, you can add one or more modifiers to the top-level modifier. A modifier can aggregate elements or other modifiers.

This section discusses the following topics:

- [Description of Element Modifiers in DTDs](#) on page 606
- [Simple Example of Aggregating Modifiers in DTDs](#) on page 607
- [More Complex Example of Aggregating Modifiers in DTDs](#) on page 608
- [Aggregating Modifiers to Allow Any Order and Any Number in DTDs](#) on page 608

Description of Element Modifiers in DTDs

[Table 78](#) describes the available modifiers:

Table 78. Element Modifiers

<i>Modifier</i>	<i>Description</i>	<i>Indicator in DTD</i>
Optional	This element can appear once or not at all. (0 or 1)	Question mark (?)
Zero or more	This element is optional and repeatable. (0, 1, or more)	Asterisk (*)
One or more	This element is required and repeatable. (1 or more)	Plus sign (+)

Table 78. Element Modifiers

<i>Modifier</i>	<i>Description</i>	<i>Indicator in DTD</i>
Choice	Exactly one of the specified subelements must appear.	Vertical bar ()
Sequence	If no other modifiers are defined on the Sequence modifier, each subelement in this element must appear exactly once. In other words, it is required. Also, the subelements must appear in the order in which they are specified in the referencing element. You can define other modifiers on the Sequence modifier. In this way, you can specify that some subelements are optional, some appear zero or more times, and some appear one or more times.	Comma (,)

Simple Example of Aggregating Modifiers in DTDs

Suppose you want a `book` element to always contain exactly one `title` element and any number of `author` elements. The `title` and `author` elements contain only raw data. To accomplish this, you would perform steps that generate the following tree representation:

```

book
  Sequence
    title
    One or More
    author
title
  Zero or More
  #PCDATA
author
  Zero or More
  #PCDATA
    
```

In the `book` element definition, `Sequence` modifies `book` and `One or More` modifies `Sequence`. Because the `title` element immediately follows the `Sequence` modifier, the default occurrence rule is assumed. That is, the `title` element must appear exactly once. In the **Text** view of the DTD, the definition for the `book` element is as follows:

```
<!ELEMENT book (title, (author)+)>
```

More Complex Example of Aggregating Modifiers in DTDs

Following is a more complicated example. Suppose you want book elements to include

- Exactly one title
- Either an author or an editor, but it is okay if neither appear
- Zero or more paragraphs

To accomplish this, you would perform steps that generate the following tree representation:

```
book
  Sequence
    title
    Optional
      Choice
        author
        editor
    Zero or More
      paragraph
```

In the **Text** view of the DTD, the definition for the book element is as follows:

```
<!ELEMENT book (title, (author|editor)?, paragraph*)>
```

Aggregating Modifiers to Allow Any Order and Any Number in DTDs

The **Choice** modifier specifies that only one of the specified elements can appear in an instance document. However, if you specify the **Zero or More** modifier and then the **Choice** modifier, the result is that the specified elements can appear in any order and each element can appear any number of times.

The text for such an element definition is as follows:

```
<!ELEMENT A (B|C|D)*>
```

The tree representation is as follows:

```
A
  Zero or More
  Choice
    B
    C
    D
```

This allows an A element to contain

- Zero, one, or more B elements
- Zero, one, or more C elements
- Zero, one, or more D elements

Furthermore, the contained elements can be in any order.


Defining Elements in DTDs

You can define an element in the **Text** or **Tree** tab.

In the **Text** tab, you enter the text that defines your element and describes its structure. For example, to define a Catalog element that can contain one or more Publisher elements, followed by zero or more Thread elements, followed by one or more Book elements, you would enter the following:

```
<!ELEMENT Catalog ((Publisher)+,((Thread)*,(Book)+))>
```

When you define elements in the **Text** tab, you must know the syntax and keywords for what you want to define. This information is publicly available on the World Wide Web. Stylus Studio documentation does not include instructions for defining a DTD in the **Text** tab. For DTD information, see, <http://www.w3.org/TR/REC-xml>.

When you use Stylus Studio, it is easier to define an element in the Tree tab. In the **Tree** tab, you click **New Element Definition** , and Stylus Studio takes care of the syntax and keywords. In the **Tree** tab, definition of an element requires that you

1. Create the element by specifying its name. To do this, see “[Defining Elements in the DTD Tree Tab](#)” on page 610, which is the first topic in this section.
2. Define the structure of the element by specifying modifiers, defining where raw data is allowed, and adding references to other elements. To help you do this, this section discusses the following topics:

- [Specifying That an Element Can Have an Attribute in DTDs](#) on page 611
- [Specifying That an Element is Required in DTDs](#) on page 611
- [Specifying That an Element is Optional in DTDs](#) on page 612
- [Specifying That Multiple Instances of An Element Are Allowed in DTDs](#) on page 613
- [Specifying That An Element Can Contain One of a Group of Elements in DTDs](#) on page 615
- [Specifying That an Element Can Contain One or More Elements in DTDs](#) on page 616
- [Specifying That an Element Can Contain Data in DTDs](#) on page 618
- [Moving, Renaming, and Deleting Elements in DTDs](#) on page 618

Defining Elements in the DTD Tree Tab

In the DTD editor, if the **Tree** view is not visible, click the **Tree** tab at the bottom of the window.

◆ **To create an element in the Tree tab:**


1. Click the **DTD** node at the top of the tree.
2. In the tool bar on the left, click **New Element Definition** . Stylus Studio displays a field for the new element at the end of the current contents of the DTD.
3. Type the name of the new element and press Enter. Stylus Studio displays the properties for the new element in the **Properties** window. If the **Properties** window is not visible, select **View > Properties** from the Stylus Studio menu bar. For example, suppose you specified `title` as the name of the new element. Your new element has the following properties:

Table 79. Element Properties


<i>Property</i>	<i>Value</i>
Type	Element
Name	<code>title</code>
Content Model	Empty

4. To change the value of a property, double-click the current value. For information about the properties that each element can have, see [“About Node Properties in DTDs”](#) on page 622.

After you create an element, you must define the structure of the contents of the element. The rest of the topics in this section provide information on how to define structure.

Specifying That an Element Can Have an Attribute in DTDs



◆ In the DTD Tree tab, to specify that an element can have an attribute:

1. Click the name of the element that you want to have an attribute.
2. In the menu bar on the left, click **New Attribute** .
3. Type the name of the attribute and press Enter.

Specifying That an Element is Required in DTDs

You specify that an element is required when you add a reference to that element in another element.

◆ In the Tree tab, to specify that an element is required:

1. Define the element that you want to be required. See [“Defining Elements in the DTD Tree Tab”](#) on page 610.
2. Create the element that contains the element that you want to be required. This is the container element.
3. Click the container element name.
4. In the tool bar on the left, click **New Modifier** . Stylus Studio displays a drop-down menu.
5. Double-click **Sequence**.
6. If the required element can appear only once, skip this step. If the required element can appear more than once, click **New Modifier** and double-click **One or More** in the pop-up menu.
7. With the modifier highlighted, click **New Reference to Element**  in the tool bar on the left.

8. Enter the name of the element that you want this element to reference.
After you add a reference to an element, you might want to check the definition of the referenced element. To do this, right-click the reference. In the shortcut menu, click **Go To Definition**. Stylus Studio moves the focus to the definition of the referenced element.

For example, suppose the `title` element is required, and that it is relevant only in the context of a `book` element. When you define the `book` element, you specify that it contains the `title` element. If you specify only the Sequence modifier, the occurrence default is assumed. The occurrence default is that there must be exactly one of the contained element. In other words, the `title` element is required and there can be only one. In this case, the definition of the `book` element is as follows:

```
<!ELEMENT book (title)>
```

The tree representation looks like this:

```
book
  Sequence
    title
```

It is also possible for an element to be required and for more than one to be allowed. Suppose the `book` element must also contain at least one `author` element, but it can contain more than one `author` element. The definition of the `book` element is as follows:

```
<!ELEMENT book (title, author+)>
```



The tree representation looks like this:

```
book
  Sequence
    title
    One or More
      author
```

Specifying That an Element is Optional in DTDs

You specify that an element is optional when you add a reference to that element in another element. When an element is optional, it means that there can be one or none. If you want to specify that there can be none, one, or more, use the **Zero or More** modifier. See [“Specifying That Multiple Instances of An Element Are Allowed in DTDs”](#) on page 613.

◆ **In the Tree tab, to specify that an element is optional:**

1. Define the element that you want to be optional. See “[Defining Elements in the DTD Tree Tab](#)” on page 610.
2. Create the element that contains the element that you want to be optional. This is the container element.
3. Click the container element name.
4. In the tool bar on the left, click **New Modifier** . Stylus Studio displays a drop-down menu.
5. If the container element can contain only the optional element, skip this step. If the container element can contain more than one element, click **Sequence**.
6. Click **New Modifier**.
7. In the pop-up menu that appears, double-click **Optional**.
8. In the tool bar on the left, click **New Reference to Element**  and enter the name of the optional element. If the container element can contain additional optional elements, repeat this step for each one.

After you add a reference to an element, you might want to check the definition of the referenced element. To do this, right-click the reference. In the shortcut menu, click **Go To Definition**. Stylus Studio moves the focus to the definition of the referenced element.



Specifying That Multiple Instances of An Element Are Allowed in DTDs

You specify that multiple instances of an element are allowed when you add a reference to that element in another element. When multiple instances of an element are allowed, you specify that there can be either

- None, one, or more
- One or more

◆ **In the Tree tab, to specify that there can be multiple instances of an element:**

1. Define the element that can appear multiple times. See “[Defining Elements in the DTD Tree Tab](#)” on page 610.
2. Define the element that contains the element that can appear multiple times. This is the container element.

3. Click the container element name.
4. In the left tool bar, click **New Modifier** . Stylus Studio displays a drop-down menu.
5. If the container element can contain only one type of element, skip this step. If the container element can contain more than one type of element, double-click **Sequence**, and then click **New Modifier**.
6. Double-click **Zero or More** to allow the container element to contain zero, one, or more instances of an element. Or, double-click **One or More** to allow the container element to contain one or more instances of an element.
7. In the left tool bar, click **New Reference to Element**  and enter the name of the element that can appear multiple times. If the container element can contain additional types of elements, repeat this step for each one that can appear multiple times.

After you add a reference to an element, you might want to check the definition of the referenced element. To do this, right-click the reference. In the shortcut menu, click **Go To Definition**. Stylus Studio moves the focus to the definition of the referenced element.

Suppose there are some elements that can appear zero, one, or more times, and there are other elements that can appear one or more times. The tree representation for this might look like the following:

```
book
  Sequence
    One or More
      Author
      Format
    Zero or More
      Award
      Review
```

In this example, an instance document must contain these elements in the following order:

```
author
format
award
review
```

Suppose you want them in the following order:


```
review
format
award
author
```


In this case, the tree representation would look like this:

```
book
  Sequence
    Zero or More
      Award
    One or More
      Format
    Zero or More
      Review
    One or More
      Author
```

Specifying That An Element Can Contain One of a Group of Elements in DTDs

You might want to define an element that contains one element out of a group of elements. For example, you might want an `InventoryNumber` element to contain a `book`, `magazine`, or `newsletter` element.

- ◆ **In the Tree tab, to define an element that contains one of a group of elements:**
 1. Define the elements that your new element can contain. See “[Defining Elements in DTDs](#)” on page 609.
 2. Define the element that you want to contain another element. This is the container element.
 3. Click the container element name.
 4. In the left tool bar, click **New Modifier** . Stylus Studio displays a drop-down menu.
 5. Double-click **Choice**.

6. For each element in the group of elements from which one element can appear:
 - a. Click **New Reference to Element**  in the left tool bar.
 - b. Type the name of the element and press Enter.

After you add a reference to an element, you might want to check the definition of the referenced element. To do this, right-click the reference. In the shortcut menu, click **Go To Definition**. Stylus Studio moves the focus to the definition of the referenced element.


When the XSLT processor validates an instance document against this DTD, it ensures that each instance of the new element you just defined contains exactly one of the referenced elements.


The tree representation for an `InventoryNumber` element that can contain a `book`, `magazine`, or `newsletter` element would look like the following:

```
InventoryNumber
  Choice
    book
    magazine
    newsletter
```

Specifying That an Element Can Contain One or More Elements in DTDs

Often, you want an element to contain a sequence of elements. Some of these elements might be required, some might be optional, and some might be able to occur more than once. There might even be a group of elements in which only one can appear.

- ◆ **In the Tree tab, to define an element that contains a sequence of elements:**
 1. Define the elements that you want your new element to contain. See [“Defining Elements in DTDs”](#) on page 609.
 2. Define the element that contains the sequence of elements. This is the container element.
 3. Click the container element name.
 4. In the left tool bar, click **New Modifier** . Stylus Studio displays a drop-down menu.
 5. Double-click **Sequence**.

6. To add required elements to the container element, click **New Reference to Element**  in the left tool bar and enter the name of the required element. Do this for each required element. You can change the order later.

At this point, you can add

- Optional elements
- Elements that can appear one or more times
- Elements that can appear zero, one, or more times
- Elements that belong to a group in which only one element in the group can be present

The procedure is the same for these modifiers. The only difference is the modifier you select. For example, following are the instructions for adding optional elements:

1. In the DTD editor, click the **Sequence** modifier.
2. In the left tool bar, click **New Modifier**.
3. Double-click **Optional**.
4. For each optional element, click **New Reference to Element** and enter the name of the optional element. This works only if you want all optional elements to be consecutive. If you want optional elements to be interspersed with required elements or elements that can appear one or more times, you must perform steps 1 through 4 for each element.

In an instance document, the contained elements must appear in the order in which they are specified in the DTD.

◆ **To modify the order:**

1. Click the modifier for the element you want to move.
2. Click the up or down arrow repeatedly until the element is where you want it to be.



To move a required element that can appear only once, click its name and then use the up and down arrows.

Alternative: Right-click the item you want to move. Select **Move Up** or **Move Down** from the shortcut menu.



Specifying That an Element Can Contain Data in DTDs

To specify that an element can contain raw data, you must first define the element. See [Defining Elements in the DTD Tree Tab](#) on page 610.

◆ **In the Tree tab, to specify that an element can contain data:**


1. Click the element you want to contain data.
2. In the left tool bar, click **New Modifier** . Stylus Studio displays a drop-down menu.
3. Double-click **Zero or More**.
4. In the left tool bar, click **Add \$PCDATA** .

Moving, Renaming, and Deleting Elements in DTDs

To move an element definition or a reference to an element, in the **Tree** tab, click the name of the element or the modifier for the reference. Then click **Move Up**  or **Move Down**  repeatedly until the element or reference is where you want it to be.

Alternative: Right-click the item you want to move. Select **Move Up** or **Move Down** from the shortcut menu that appears.

To rename an element or attribute, right-click it and select **Rename** from the shortcut menu that appears. Type the new name and press Enter.

Alternative: Click **Change Name** .

To delete a node in the DTD, right-click the node you want to delete. In the shortcut menu that appears, click **Delete**.

Alternative: Click **Delete Node** .

Defining General Entities and Parameter Entities in DTDs

In DTDs, an entity allows you to define a symbol for a value. In the **Tree** view, you can define general entities and parameter entities. The value of a general entity can be just about anything. It can be

- A short string that represents a longer string
- A way to include another marked-up file

- A reference to a graphical image
- A placeholder for some non-XML data or an expression that needs special formatting

General entities are useful for things that change often, such as the name of a product in development. An entity allows you to change the value in one place and have the corrected value appear everywhere it is needed.

See also “[Description of Entity and Parameter Entity Properties in DTDs](#)” on page 625.

When you define a general entity, you specify a symbol that you can use in instance documents. When the XML parser finds a reference to a general entity, it replaces the symbol with the value you specified when you defined the general entity.

When you define a parameter entity, you specify a symbol that you can use elsewhere in the DTD. Again, when the XML parser finds a reference to a parameter entity, it replaces the reference with the value you specified when you defined the parameter entity.

In a DTD, the definition of an entity must appear before a reference to that entity. Therefore, it is good practice to put all entity declarations at the beginning of a DTD.



This section discusses the following topics:

- [Steps for Defining Entities in DTDs](#) on page 619
- [General Entity Example in a DTD](#) on page 620
- [Parameter Entity Example in a DTD](#) on page 621

Steps for Defining Entities in DTDs

The procedures for defining general entities and parameter entities are almost the same.

◆ To define an entity in the **Tree** tab:

1. Click the **DTD** node.
2. In the left tool bar, click **New Entity**  or **New Parameter Entity** .
3. Type the name of the new entity and press Enter. Stylus Studio displays the properties for the new entity in the **Properties** window. If the **Properties** window is not visible, select **View > Properties** from the Stylus Studio menu bar to display it.
4. In the **Properties** window, check the value of the **Location** property.

If you want to define the value for this entity in this DTD, the value should be **Internal**. Otherwise, the value should be **External**. If you need to change the value of the

Location property, double-click its current value. In the drop-down menu that appears, double-click the new value.

5. If the value of the **Location** property is **External**, skip this step. If the value of the **Location** property is **Internal**, double-click the **Value** field and enter the value of the entity. Definition of your entity is complete. You do not perform the remaining steps in this procedure.
6. If the value of the **Location** property is **External**, specify a value for **System ID**. Double-click the field to enter a value. The value of **System ID** is a path to a file. It can be a URL or a file system path.

Although you can also specify a value for the **Public ID** property, Stylus Studio ignores any value you specify. A **Public ID** is a string that some parsers can resolve to an address, which they then use to locate a file. Stylus Studio does not have this capability.

7. If you are defining a parameter entity, you are done. If you are defining a general entity, check the value of the **Parsed** property. If necessary, double-click the value of the **Parsed** property to change it. The value of the **Parsed** property indicates whether the value of the entity is parsed XML. For example, if the entity refers to an image file, you do not want Stylus Studio to try to parse it.

General Entity Example in a DTD

Suppose you define the `shopname` general entity as an internal entity with the value `Most Excellent Book Store of Tokyo`. In the **Text** view of the DTD, this appears as follows:

```
<!ENTITY shopname "Most Excellent Book Store of Tokyo">
```

In an instance document, when the XML parser finds `&shopname;`, it replaces it with `Most Excellent Book Store of Tokyo`.

Parameter Entity Example in a DTD

Suppose you define the `invoice` parameter entity as an internal entity as follows:

```
<!ENTITY % customer "name, street, city, state, zipcode">
```

The percent sign (%) after the `ENTITY` keyword indicates that this is a parameter entity. Later in the DTD, you can reference this parameter entity as follows:

```
<!ELEMENT invoice (%customer;, item, price, date)>
```


When this DTD is processed, it is as if you had specified the following:

```
<!ELEMENT invoice (name, street, city, state, zipcode, item, price, date)>
```

Inserting White Space in DTDs

Suppose you define some elements in the **Tree** tab. If you click the **Text** tab, you see that your DTD is on one long line. To make your DTD more readable, you can insert white space between elements. You cannot insert white space between the nodes that define an element.


◆ **In the Tree tab, to insert white space:**

1. Click the **DTD** node at the top of the schema.
2. In the tool bar on the left, click **New Text** .
3. Type a space and press Enter.
4. Click the up arrow to move the space to the desired location.

Adding Comments to DTDs

In a DTD, comments are useful for organizing the contents and clarifying the various parts of a DTD. A comment can appear between element, entity, or white space nodes. You can insert a comment in the middle of an element definition.

◆ **In the Tree tab, to insert a comment:**

1. Click the **DTD** node at the top of the schema.
2. In the tool bar on the left, click **New Comment** .

3. Type your comment and press Enter.
4. Click the up arrow to move the comment to the desired location.

About Node Properties in DTDs

Each node in a DTD is associated with one or more properties. Every node has a **Type** property. The properties associated with a node vary according to the value of the **Type** property. Stylus Studio supports the following values for the **Type** property of a node in a DTD:

- Element
- Attribute
- DTD Modifier
- PCDATA
- Entity
- Parameter Entity
- Text
- Comment

To determine the properties for a particular node in a DTD, click the node. Stylus Studio displays the properties in the **Properties** window. If the **Properties** window is not visible, select **View > Properties** from the Stylus Studio menu bar.

To change a property, double-click the property value in the **Properties** window. Enter the new value or, if a drop-down menu appears, double-click the value you want. Any changes you make in the **Properties** window are immediately reflected in the **Tree** and **Text** views. You cannot change the type property of a node.

The remainder of this section discusses the following topics:

- [Description of Element Properties in DTDs](#) on page 623
- [Description of Attribute Properties in DTDs](#) on page 623
- [Description of Entity and Parameter Entity Properties in DTDs](#) on page 625

Description of Element Properties in DTDs

An element has three properties: Type, Name, and Content Model. The Name property is a string that identifies the element. The Content Model property describes the allowed contents for the element. [Table 80](#) describes the possible values of the Content Model property for Element nodes:

Table 80. Element Property Descriptions

<i>Value of Content Model Property</i>	<i>Description</i>
Empty	This element can contain only attributes.
Element Only	This element can contain attributes and specified elements. It cannot directly contain raw data.
Mixed	This element can contain attributes, specified elements, and raw data.
Any	This element can contain attributes, any elements defined in this DTD, and raw data.

Description of Attribute Properties in DTDs

[Table 81](#) shows the properties that an attribute can have. It also provides the possible values, and a description for each property.

Table 81. Attribute Property Descriptions

<i>Property</i>	<i>Allowable Values</i>	<i>Description</i>
Type	Attribute	All attribute nodes have this type.
Name	String	Identifier for the particular attribute.
Restrictions	Fixed	The attribute is required and it must always have the value specified by the Default property. You must always explicitly specify this attribute.
	Implied	The attribute is optional. There is no default value.
	Optional	The attribute is optional. If you do not specify it, the XML parser uses the value of the Default property.

Table 81. Attribute Property Descriptions

<i>Property</i>	<i>Allowable Values</i>	<i>Description</i>
	Required	The element must always explicitly specify this attribute and assign a value to it.
Content Type	CDATA	The attribute value can contain any valid character data. It is a text string.
	Entity	The attribute value is the name of an entity defined in the DTD.
	Entities	The attribute value is a space-separated list of entities that are defined in the DTD.
	Enumerated	The attribute value is one of a set of specified values. When the value of the Content Type property is Enumerated, the attribute has an additional property: Allowed Values. Specify the allowed values in a space-separated list.
	ID	The attribute value is a unique name within the DTD.
	IDREF	The attribute value is an ID that is defined in the DTD.
	IDREFs	The attribute value is a space-separated list of IDs that are defined in the DTD.
	NMToken	The attribute value is a valid XML name that is composed of letters, numbers, hyphens, underscores, and colons.

Table 81. Attribute Property Descriptions

<i>Property</i>	<i>Allowable Values</i>	<i>Description</i>
	NMTokens	The attribute value is a space-separated list of name tokens.
	Notation	The name of a notation specified in the DTD. The notation describes a non-XML data format, such as those used for image files. When the value of the Content Type property is Notation, the attribute has an additional property: Allowed Values. Specify the allowed values in a space-separated list.

Description of Entity and Parameter Entity Properties in DTDs

Table 82 shows the properties that an entity or parameter entity can have. It also provides the possible values, and a description for each property.

Table 82. Entity and Parameter Entity Property Descriptions

<i>Property</i>	<i>Allowable Values</i>	<i>Description</i>
Type	Entity Parameter Entity	All entity nodes have this type.
Name	String	Identifier for this entity.
Location	External or Internal	An external location indicates that the value of the entity is in a file that is outside the DTD file. An internal location indicates that the value of the entity is defined in the Value property of this entity node.
Value	String	If the value of the Location property is Internal, this property specifies the value of the entity. If the value of the Location property is External, you cannot specify this property.

Table 82. Entity and Parameter Entity Property Descriptions

<i>Property</i>	<i>Allowable Values</i>	<i>Description</i>
Public ID	String	String that some parsers can resolve to a file location. Stylus Studio ignores any value you specify.
System ID	String	Path or URI for a file that contains the value of the entity.
Parsed	True or False	Indicates whether the entity value is parsed XML. A parameter entity does not have this property.

Associating an XML Document with an External DTD

To associate an XML document with an external DTD, add a DOCTYPE element to the beginning of your XML document. The DOCTYPE element should be immediately after the XML declaration element. The format of the DOCTYPE element is

```
<!DOCTYPE root_element_name SYSTEM "path_to_dtd">
```

Replace *root_element_name* with the name of the root element in your XML document.

Replace *path_to_dtd* with the path for the DTD you want your document to use.

Moving an Internal DTD to an External File

◆ **To move an internal DTD to an external file:**

1. In the **Text** tab of the DTD editor, in the DOCTYPE element, select only the text inside the brackets [].
2. Cut the text.
3. Select **File > New > DTD Schema** from the menu bar.
4. Paste the text in the new DTD schema file that Stylus Studio displays.
5. Save the file. You might want to save the DTD in the same directory as the XML document that uses it.

6. In your XML document in the **Text** tab of the DTD editor, remove the brackets and insert the following in their place:

```
SYSTEM "schema_file_path"
```

The path you specify can be the relative or absolute path of the DTD file you just saved. This path must be in quotation marks.

Chapter 9 **Writing XPath Expressions**

The XML Path Language (XPath) allows you to query an XML document using XPath expressions. An XPath expressions returns a well-formed XML node-list or an XPath value object. Stylus Studio supports the November 2005 W3C XPath 2.0 Candidate Recommendation. XPath 2.0 is a superset of XPath 1.0.

This section discusses the following topics:

- [“About the XPath Processor”](#) on page 630
- [“Using the XPath Query Editor”](#) on page 632
- [“Sample Data for Examples and Practice”](#) on page 639
- [“Getting Started with Queries”](#) on page 643
- [“Specifying the Nodes to Evaluate”](#) on page 656
- [“Handling Strings and Text”](#) on page 670
- [“Specifying Boolean Expressions and Functions”](#) on page 677
- [“Specifying Number Operations and Functions”](#) on page 680
- [“Comparing Values”](#) on page 683
- [“Finding a Particular Node”](#) on page 688
- [“Obtaining a Union”](#) on page 695
- [“Obtaining Information About a Node or a Node Set”](#) on page 697
- [“Using XPath Expressions in Stylesheets”](#) on page 701
- [“Accessing Other Documents During Query Execution”](#) on page 705
- [“XPath Quick Reference”](#) on page 707

About the XPath Processor

Stylus Studio supports the November 2005 W3C XPath 2.0 Candidate Recommendation. XPath 2.0 is a superset of XPath 1.0.

As an overview of the XPath processor, this section provides the following information:

- [Where You Can Use XPath Expressions](#) on page 630
- [About XPath](#) on page 630
- [Benefits of XPath](#) on page 631
- [Internationalization](#) on page 632
- [Restrictions on Queries](#) on page 632

For additional information about XPath see <http://www.w3.org/TR/xpath20>.

Where You Can Use XPath Expressions

You use XPath expressions in XQuery documents and XSLT stylesheets to select the nodes you want to transform and query. For example, you can specify queries as values of `match` and `select` attributes in stylesheets. You can use XPath 1.0 expressions in XQuery and XSLT 1.0, and XPath 2.0 expressions in XSLT 2.0.

You can also query XML documents using the XPath Query Editor. Stylus Studio displays the results in the **Query Output** window, Stylus Studio displays the result of the query.

About XPath

XPath is a notation for retrieving information from a document. The information could be a set of nodes or derived values.

XPath allows you to identify parts of an XML document. In addition, a subset of XPath allows you to test whether or not a node matches a particular pattern. XPath provides Boolean logic, filters, indexing into collections of nodes, and more.

XPath is declarative rather than procedural. You use a pattern modeled on directory notation to describe the types of nodes to look for. For example, `book/author` means find all author elements that are contained in book elements.

XPath provides a common syntax for features shared by Extensible Stylesheet Language Transformations (XSLT) and XQuery. XSLT is a language for transforming XML documents into XML, HTML, or text. XQuery builds on XPath and is a language for extracting information from XML documents.

The basic syntax for XPath mimics the Uniform Resource Identifier (URI) directory navigation syntax. However, the syntax does not specify navigation through a physical file structure. The navigation is through elements in the XML tree.

Benefits of XPath

XPath is designed for XML documents. It provides a single syntax that you can use for queries, addressing, and patterns. XPath is concise, simple, and powerful. XPath has many benefits, as follows:

- Queries are compact.
- Queries are easy to type and read.
- Syntax is simple for the simple and common cases.
- Query strings are easily embedded in programs, scripts, and XML or HTML attributes.
- Queries are easily parsed.
- You can specify any path that can occur in an XML document and any set of conditions for the nodes in the path.
- You can uniquely identify any node in an XML document.
- Queries return any number of results, including zero.
- Query conditions can be evaluated at any level of a document and are not expected to navigate from the top node of a document.
- Queries do not return repeated nodes.
- For programmers, queries are declarative, not procedural. They say *what* should be found, not *how* it should be found. This is important because a query optimizer must be free to use indexes or other structures to find results efficiently.
- XPath is designed to be used in many contexts. It is applicable to providing links to nodes, for searching repositories, and for many other applications.

When you define a query, keep in mind that XML data can be represented as a tree. A *tree* is a hierarchical representation of XML data. The *root node* is the top of the tree. Each element, attribute, text string, comment, and processing instruction corresponds to one node in the tree. A tree also shows the relationships among the nodes. For more information on tree structure, see [“Tree Representation of a Sample XML Document”](#) on page 640.

Internationalization

Queries can contain non-Latin characters.

Restrictions on Queries

XPath is a language for selecting existing XML data; it does not perform manipulation (like sorting) or construction of different XML structures. To perform such operations, you need to use the language that is hosting XPath – XSLT or XQuery, for example.

You cannot query non-XML data. If you query a document that does not contain XML-formatted data, Stylus Studio displays an error message that informs you that the queried text is not XML.

Using the XPath Query Editor



The XPath Query Editor is available only in Stylus Studio XML Enterprise Suite and Stylus Studio Professional Suite.

The XPath Query Editor is a dockable window that you can use to query XML documents in Stylus Studio. An example, showing an XPath expression being evaluated against `bookstore.xml` from the Example project installed with Stylus Studio, is shown here:

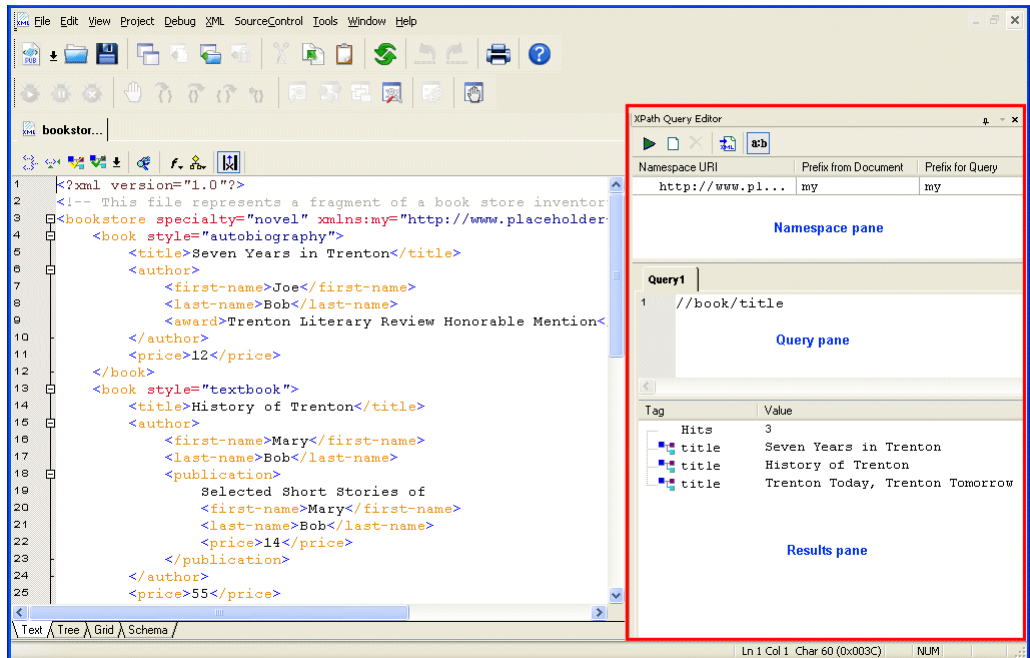


Figure 280. XPath Query Editor



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the Introduction to the XPath Query Editor video](#).

A complete list of the videos demonstrating Stylus Studio's features is here: http://www.stylusstudio.com/xml_videos.html.

Parts of the XPath Query Editor

The XPath Query Editor consists of

- A *query pane*, in which you enter the XPath expression you want to execute against the current document. The query pane allows you to enter queries with multiple lines.

The query pane supports Stylus Studio's Sense:X feature, which provides tool-tips for XPath expressions and auto-completion for XML documents associated with an XML Schema, as shown here:

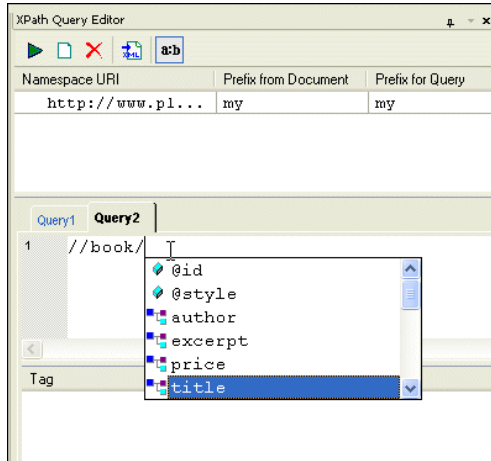



Figure 281. Sense:X Auto-completion in XPath Query Editor

- A *results pane*, which shows the results from the query after you execute it. Results are shown as a number of hits and include the type of XML tag and value of each hit. When you click on a returned node, Stylus Studio moves the cursor in the XML document to the source node for the returned node.
- A *namespace pane*, which allows you to redefine the namespace prefix to be used in the query. The namespace pane is not displayed by default, and is not shown in [Figure 280](#).

Displaying the XPath Query Editor

The XPath Query Editor is not displayed by default. In addition, it is closed when you exit the XML document with which it is associated.

◆ To display the XPath Query Editor:

- Click the **Show XPath Query Editor** button on the XML Editor tool bar (). The XPath Query Editor appears. Any previously created queries are displayed on individual tabs. If no queries have been created, a new tab, labeled *Query1*, is displayed.

Alternative:

- Select **View > XPath Query Editor** from the Stylus Studio menu.

Customizing Syntax Coloring

The XPath Query Editor uses Stylus Studio's syntax coloring, and you can change the default settings for several XPath Query Editor properties on the **Editor Format** page of the **Options** dialog box. These settings affect tokens in XPath expressions, such as errors, strings, keywords, operators, and attributes.

◆ To change syntax coloring in the XPath Query Editor:

1. Click **Tools > Options > General > Editor Format** to display the **Editor Format** page of the **Options** dialog box.
The **Options** dialog box appears.
2. Scroll the **Colors** list; fields related to the XPath Query Editor are prefixed with **XPath**.
3. Use the palette to change the color of the tokens as desired.
4. Click **OK**.

Working with XPath Queries

When you display the XPath Query Editor, an empty query is created for you. The query name, *Query1*, is displayed on a tab in the **XPath Query Editor** window. You can start typing the XPath expression on the first line in the editing pane; use the Enter key to move the cursor to a new line.

Tip Line numbers are displayed in the editing pane if you have enabled them for XML documents. Click **Tools > Options > General > Editor General** to change this setting.


You can create up to sixty-four queries for a single document; each is given the name *Query n* , where n is a unique number incremented by one. You cannot name queries.

Queries are saved with the project. Changes you make – either to query expressions, or creating and deleting queries – are saved when you save the project, and not the document.

Executing the Query

Stylus Studio executes XPath query expressions based on the cursor's current position in the document, regardless of which editor is currently active.

◆ To execute the query:


- Click the **Execute** button on the XPath Query Editor tool bar ().
- Stylus Studio processes the query and displays the result in the results pane.

Alternatives:

- Press F5.
- or
- Right-click the query panel in the XPath Query Editor and select **Execute Query** from the short cut menu.

Creating a New Query

◆ To create a new query:


- Click the **New XPath Query** button on the XPath Query Editor tool bar ().
- A new tab appears in the XPath Query Editor.

Alternative:

- Right-click the query panel in the XPath Query Editor and select **New XPath Query** from the short cut menu.

Deleting a Query

◆ To delete a query:

- Click the **Delete** button on the XPath Query Editor tool bar ().
- The tab associated with the query is removed from the XPath Query Editor. The query itself is deleted when you save the project.

Alternative:

- Right-click the query panel in the XPath Query Editor and select **New XPath Query** from the short cut menu.

Working with Query Results

Query results are shown in the XPath Query Editor in the results pane as a number of hits and include the type of XML tag and value of each hit, as shown here.

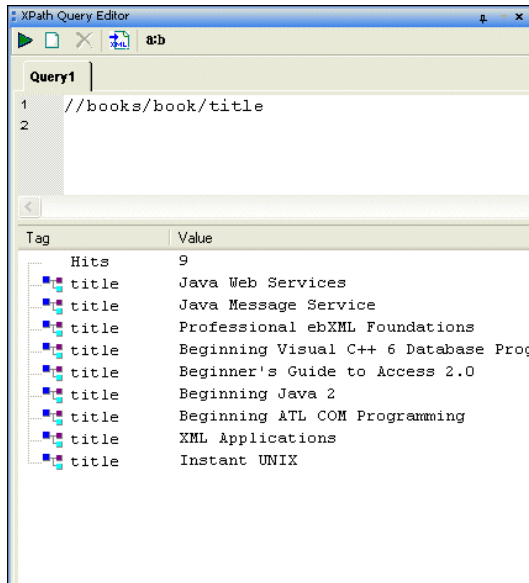


Figure 282. XPath Result Pane


When you click on a result node, Stylus Studio's back-mapping feature highlights the line in the XML document that supplied the value for the result node you selected.

Opening Query Results as a New Document

After you have executed a query, you can optionally choose to display the query result in a new XML document. If the result contains only elements, Stylus Studio creates a root element (named `<xqr:xpath-query-result>`) to ensure that the document is well-formed.

◆ To open a query result as an XML document:

Execute the query, and then

- Click the **Open result in a new XML document** button on the XPath Query Editor tool bar ()

Alternative:

- Right-click the query pane in the XPath Query Editor and select **Open result in a new XML document** from the shortcut menu.

Working with Namespaces

When the XML document declares one or more namespaces, those namespaces are displayed in the namespace pane, as shown here:

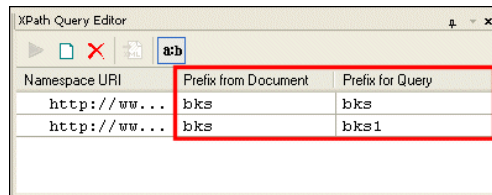


Figure 283. Namespaces Pane

In addition to the namespace URI, Stylus Studio displays the namespace prefix declared in the XML document, if any, and a prefix that is used when creating XPath query expressions.

If one namespace prefix clashes with another in the XML document, Stylus Studio renames the second prefix by adding a number to the end of the original prefix name to make it unique – for example, if two namespaces have the prefix *bks*, the second is renamed *bks1*. Similarly, if no namespace prefix has been declared, Stylus Studio creates a default namespace prefix, *ns1*. Namespace declarations in the XML document are not changed.

Namespace prefixes defined for the query


- Allow you to execute the query with the root node of the document as the current context node
- Simplify the process of entering XPath expressions in the query pane

Changes saved to the XML document are reflected in the namespace pane when you click on the namespace pane or query pane.

Viewing/Changing Namespace Prefixes

You must use the value in the **Query for Prefix** field when writing XPath query expressions; you can specify your own prefix if want to use one other than the one provided by Stylus Studio.

◆ **To change a namespace prefix:**

1. Click the **Show Namespaces** button () if the namespace pane is not already displayed.
The namespace pane shows any namespaces that have been defined for the current XML document, as well their associated namespace prefixes, if any.
2. Optionally, change the value in the **Prefix for Query** field.

Sample Data for Examples and Practice

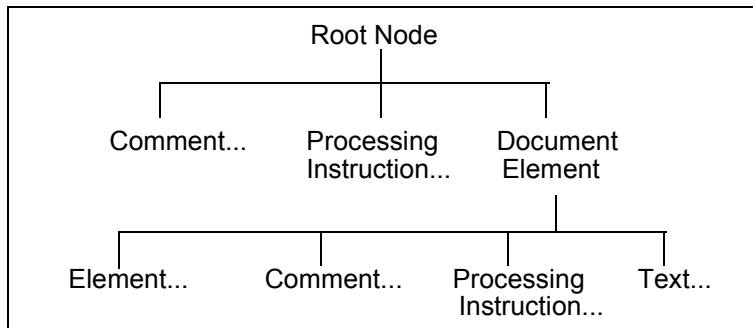
The best way to learn how to query data is to practice using XPath. To prepare you for practicing with XPath queries, this section provides a review of the basic structure of an XML document. An understanding of this structure is crucial to defining queries that return the data you want. Following the review, this section includes the XML data on which the query examples operate. The last part of this section provides instructions for running queries on sample data.

The topics in this section include

- [“About XML Document Structure”](#) on page 639
- [“A Sample XML Document”](#) on page 640
- [“Tree Representation of a Sample XML Document”](#) on page 640
- [“Steps for Trying the Sample Queries”](#) on page 643

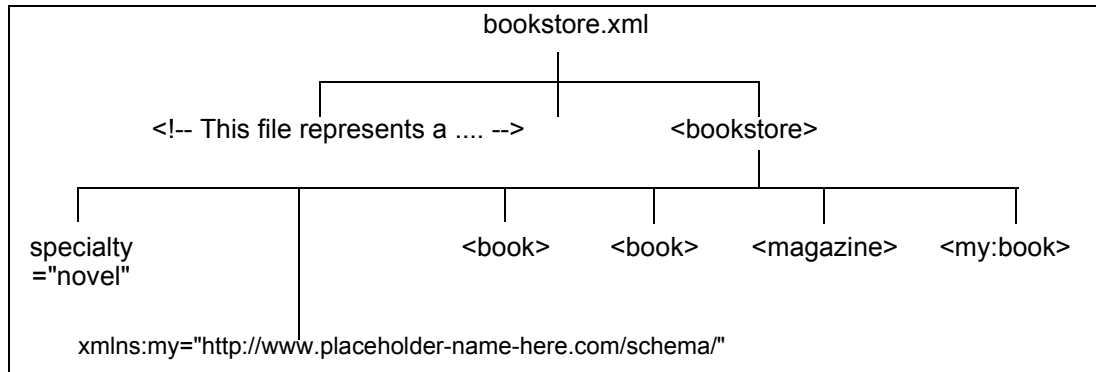
About XML Document Structure

The XPath processor operates on a tree representation of XML data that looks like the following figure:



The root node has no actual text associated with it. You can think of the file name as the root node. A document can include zero or more comments and zero or more processing instructions.

A document element is required, and there can be only one. The document element contains all elements in the document. For example:



In the preceding figure, `bookstore.xml` is the name of a file that contains XML data. There is a comment near the beginning of the document that starts with "This file represents a . . ." The document element is `bookstore`. The immediate children of `bookstore` include an attribute, a namespace declaration (not supported by Stylus Studio), three `book` elements (one is in the `my` namespace), and a `magazine` element. The `book` and `magazine` elements contain elements and attributes, which are shown in the [figure](#) that appears in "Tree Representation of a Sample XML Document" on page 640

A Sample XML Document

The examples in this section are based on the following XML data. This data is in the `bookstore.xml` file, which is in the `examples` directory of your installation directory.

Tree Representation of a Sample XML Document

When you query a document, it can be helpful to think of a tree representation of your data. A tree that represents the `bookstore.xml` document appears in [Figure 284](#) (and is

continued in [Figure 285](#)). To use Stylus Studio to view a similar tree for any XML document, open the XML document in Stylus Studio and select the **Tree** tab.

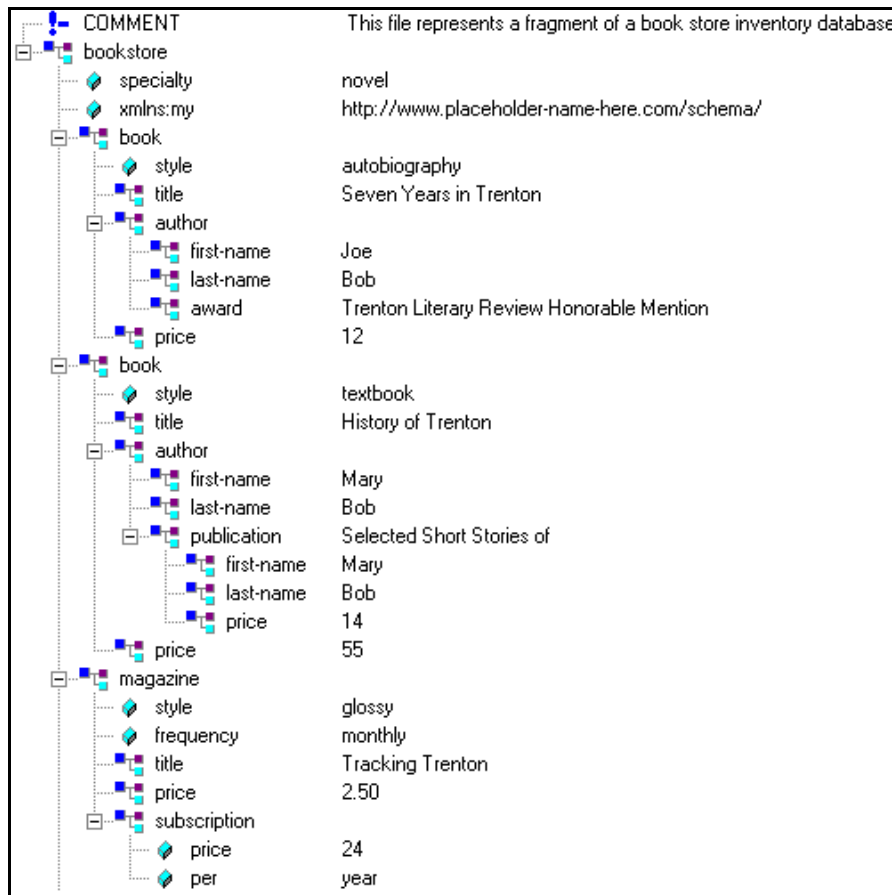


Figure 284. Tree Display of an XML Document

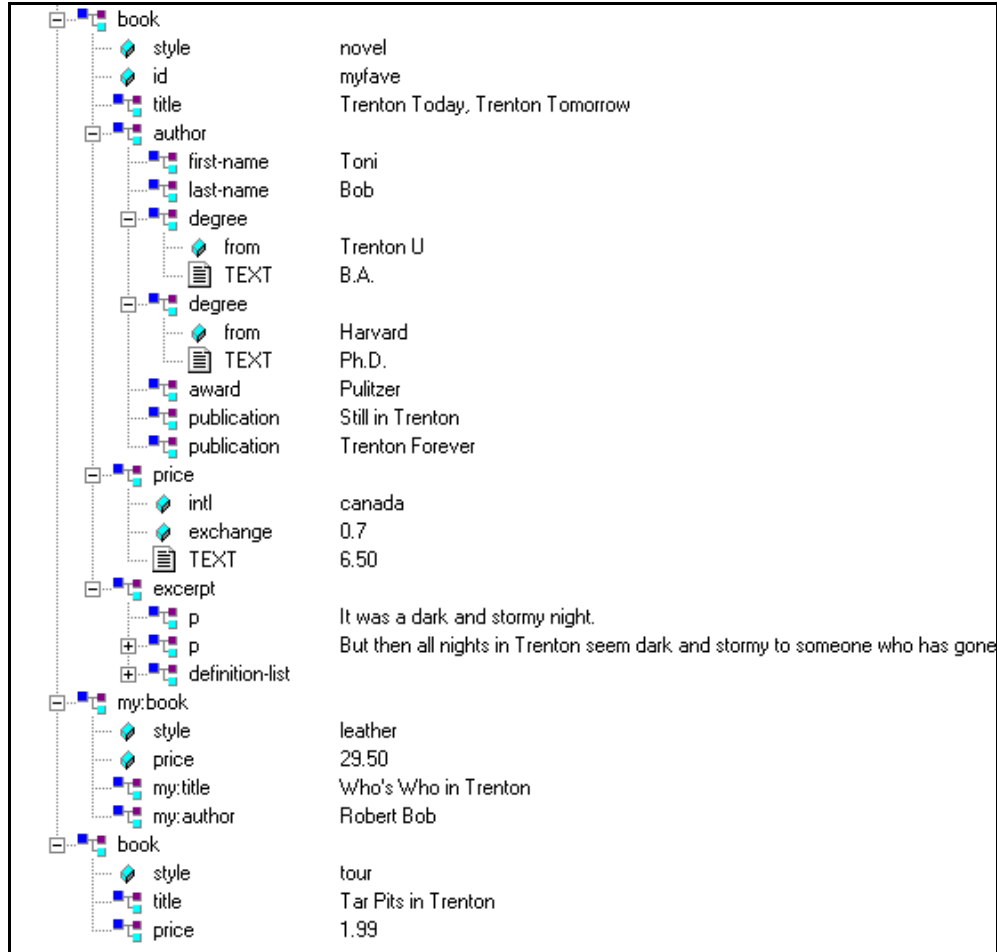


Figure 285. Tree Display of an XML Document (continued)

Steps for Trying the Sample Queries

To try the queries in this section, or any other queries you want to run on the `bookstore.xml` document, follow these instructions:

1. In Stylus Studio, open `bookstore.xml`. You can find it in the `examples` directory of your installation directory.
2. Open the XPath Query Editor if it is not already displayed. See [“Displaying the XPath Query Editor”](#) on page 634 if you need help with this step.
3. Type a query. For example: `/bookstore/book/author`.
4. Press F5, or click **Execute Query** (▶).

Stylus Studio displays the results in the **Query Output** window.

Getting Started with Queries

This section provides information to get you started using queries. It does not provide complete information about how to define a query. Instead, it provides instructions for defining typical queries you might want to run. There are numerous cross-references to later sections that provide complete information about a particular query construct.

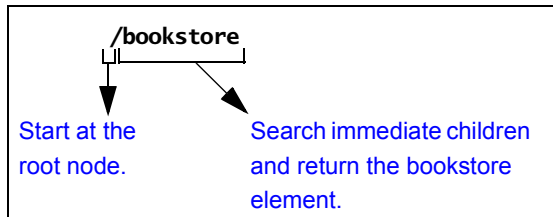
The topics discussed in this section include

- [“Obtaining All Marked-Up Text”](#) on page 644
- [“Obtaining a Portion of an XML Document”](#) on page 644
- [“Obtaining All Elements of a Particular Name”](#) on page 645
- [“Obtaining All Elements of a Particular Name from a Particular Branch”](#) on page 646
- [“Different Results from Similar Queries”](#) on page 647
- [“Queries That Return More Than You Want”](#) on page 647
- [“Specifying Attributes in Queries”](#) on page 648
- [“Filtering Results of Queries”](#) on page 649
- [“Wildcards in Queries”](#) on page 652
- [“Calling Functions in Queries”](#) on page 653
- [“Case Sensitivity and Blank Spaces in Queries”](#) on page 654
- [“Precedence of Query Operators”](#) on page 655

Obtaining All Marked-Up Text

When you query a document, you do not usually want to obtain all marked-up text. However, an understanding of queries that return all marked-up text makes it easier to define a query that retrieves just what you want.

The following figure shows a complete query (`/bookstore`) and the way the XPath processor interprets it:



This query returns the bookstore element. Because the bookstore element is the document element, which contains all elements and attributes in the document, this query returns all marked-up text.

In the query, the initial forward slash (`/`) instructs the XPath processor to start its search at the root node.

Suppose you run the following query on `bookstore.xml`:

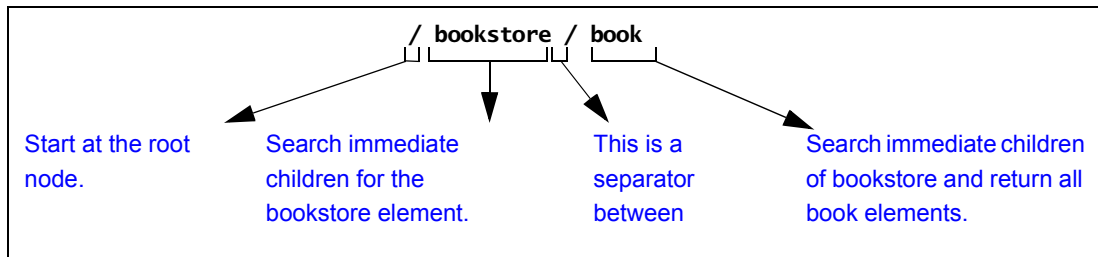
```
/book
```

This query returns an empty set. It searches the immediate children of the root node for an element named `book`. Because there is no such element, this query does not return any marked-up text. Note that this query does not return an error. The query runs successfully, but the XPath processor does not find any elements that match the query. All `book` elements are grandchildren of the root node, and the XPath processor only checks the children of the root node.

Obtaining a Portion of an XML Document

Usually, you use a query to obtain a portion of an XML document. To obtain the particular elements that you want, you must understand how to obtain an element that is a child of the document element. With this information, you can obtain any elements in the document.

The following figure shows how the XPath processor interprets the `/bookstore/book` query:



When the XPath processor starts its search at the root node, there is only one element among the immediate children of the root node. This is the document element. In this example, `bookstore` is the document element.

The query in this figure returns the `book` elements that are children of `bookstore`. This query does not return the `my:book` element, which is also a child of `bookstore`.

Now you can define queries that obtain any elements you want. For example:

```
/bookstore/book/title
```

This query returns `title` elements contained in `book` elements that are contained in `bookstore`.

Obtaining All Elements of a Particular Name

Sometimes you want all like-named elements regardless of where they are in a document. In this case, you do not need to start at the root node and navigate to the elements you want.

For example, the following query returns all `last-name` elements in any XML document:

```
//last-name
```

The double forward slash (`//`) at the beginning of a query instructs the XPath processor to start at the root node and search the entire document. In other words, the XPath processor searches all descendants of the root node.

If you perform this query on `bookstore.xml`, it returns the `last-name` elements that are children of `author` elements, and it also returns the `last-name` element that is a child of a `publication` element.

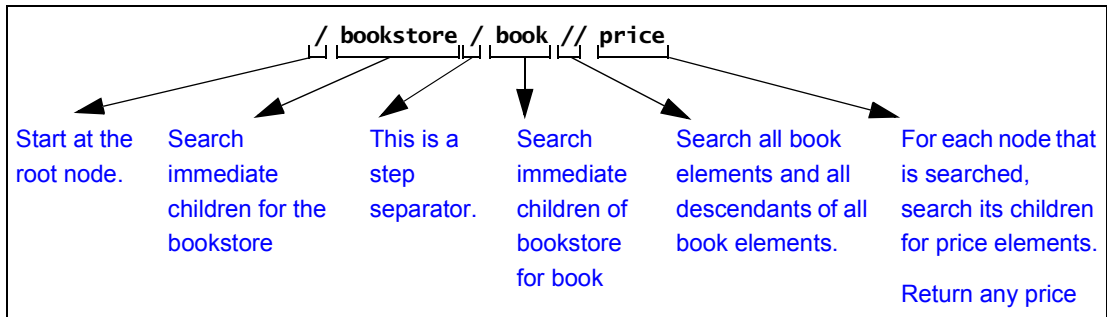
Obtaining All Elements of a Particular Name from a Particular Branch

Although sometimes you might want all like-named elements wherever they are in a document, other times you might want only those like-named elements from a particular part of the document (branch of the tree).

For example, you might want all price elements contained in book elements, but not price elements contained in magazine elements. The query is to return such a result is:

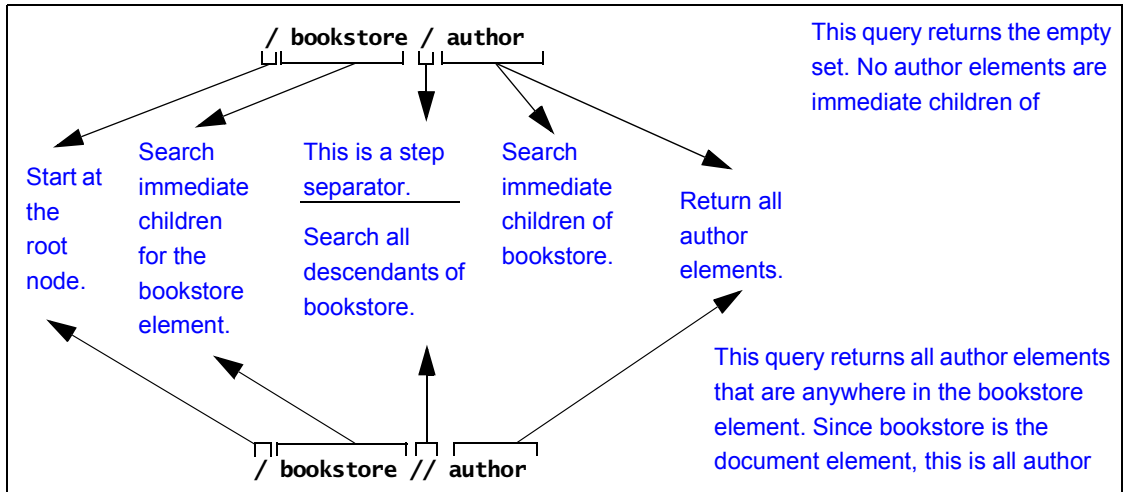
```
/bookstore/book//price
```

This query returns all price elements that are contained in book elements. Some of these price elements are immediate children of book elements. One returned price element is a great-grandchild of the second book element. The following figure shows how the XPath processor interprets this query:



Different Results from Similar Queries

Some queries can look very similar but return very different results. The following figure shows this.



Queries That Return More Than You Want

Suppose you want the titles of all the books. You might decide to define your query like this:

```
//title
```

This query does return all titles of books, but it also returns the title of a magazine. This query instructs the XPath processor to start at the root node, search all descendants, and return all `title` elements. In `bookstore.xml`, this means that the query returns the title of the magazine in addition to the titles of books. In some other document, if all titles are contained in book elements, this query returns exactly what you want.

To query and obtain only the titles of books, you can use either of the following queries. They obtain identical results. However, the first query runs faster.

```
/bookstore/book/title
//book/title
```

The first query runs faster because it uses the `child` axis, while the second query uses the `descendant-or-self` axis. In general, the simpler axes, such as `child`, `self`, `parent`, and

ancestor, are faster than the more complicated axes, such as `descendent`, `preceding`, `following`, `preceding-sibling`, and `following-sibling`. This is especially true for large documents. Whenever possible, use a simpler axis.

Specifying Attributes in Queries

To specify an attribute name in a query, precede the attribute name with an at sign (`@`). The XPath processor treats elements and attributes in the same way wherever possible. For example:

```
//@style
```

This query returns the `style` attributes associated with the `magazine`, the three `books`, and the `my:book` element. That is, it returns all the `style` attributes in the document. It does not return the elements that contain the attributes.

Following is another query that includes an attribute:

```
/bookstore/book/@style
```

This query returns the three `style` attributes for the three `book` elements.

The following query returns the `style` attribute of the context node:

```
@style
```

If the context node does not have a `style` attribute, the result set is empty.

The next query returns the `exchange` attribute on `price` elements in the current context:

```
price/@exchange
```

Following is an example that is not valid because attributes cannot have subelements:

```
price/@exchange/total
```

Following is a query that finds the `style` attribute for all `book` elements in the document:

```
//book/@style
```

Restrictions

Attributes cannot contain subelements. Consequently, you cannot apply a path operator to an attribute. If you try to, you receive a syntax error.

Attributes are inherently unordered. Consequently, you cannot apply a position number to an attribute. If you try to, you receive a syntax error.

Attributes and Wildcards

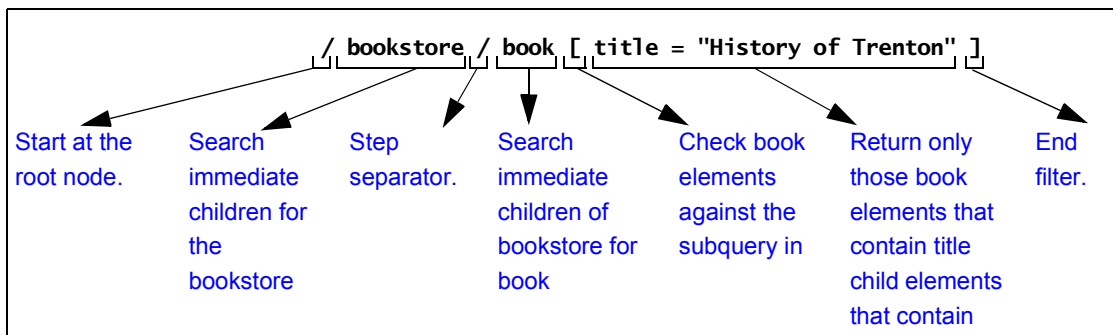
You can use an at sign (@) and asterisk (*) together to retrieve a collection of attributes. For example, the following query finds all attributes in the current context:

```
@*
```

Filtering Results of Queries

Sometimes you want to retrieve only those elements that meet a certain condition. For example, you might want information about a particular book. In this case, you can include a filter in your query. You enclose filters in brackets ([]).

The following figure shows how the XPath processor interprets a query with a filter:



This query checks each book element to determine whether it has a `title` child element whose value is "History of Trenton". If it does, the query returns the book element. Using the sample data, this query returns the second book element.

The following topics provide details about filters:

- [“Quotation Marks in Filters”](#) on page 650
- [“More Filter Examples”](#) on page 650
- [“How the XPath Processor Evaluates a Filter”](#) on page 651

- [“Multiple Filters”](#) on page 651
- [“Filters and Attributes”](#) on page 652

Quotation Marks in Filters

Suppose you define the following filter:

```
[title="History of Trenton"]
```

If you need to specify this filter as part of an attribute value, use single quotation marks instead of double quotation marks. This is because the attribute value itself is (usually) inside double quotation marks. For example:

```
<xsl:value-of select="/bookstore/book[title='History of Trenton']">
```

Strings within an expression may contain special characters such as [, {, &, ', /, and others, as long as the entire string is enclosed in double quotes ("). When the string itself contains double quotes, you may enclose it in single quotes ('). When a string contains both single and double quotes, you must handle these segments of the string as if they were individual phrases, and concatenate them.

More Filter Examples

Following is another example of a query with a filter clause. This query returns book elements if the price of the book is greater than 25 dollars:

```
/bookstore/book[price > 25]
```

The next query returns author elements if the author has a degree:

```
//author[degree]
```

The next query returns the date attributes that match "3/1/00":

```
//@date[.="3/1/00"]
```

The next query returns manufacturer elements in the current context for which the `rwdrive` attribute of the `model` is the same as the `vendor` attribute of the manufacturer:

```
manufacturer[model/@rwdrive = @vendor]
```


How the XPath Processor Evaluates a Filter

You can apply constraints and branching to a query by specifying a filter clause. The filter contains a query, which is called the *subquery*. The subquery evaluates to a Boolean value, or to a numeric value. The XPath processor tests each element in the current context to see if it satisfies the subquery. The result includes only those elements that test true for the subquery.

The XPath processor always evaluates filters with respect to a context. For example, the expression `book[author]` means for every book element that is found in the current context, determine whether the book element contains an author element. For example, the following query returns all books in the current context that contain at least one excerpt:

```
book[excerpt]
```

The next query returns all titles of books in the current context that have at least one excerpt:

```
book[excerpt]/title
```

Multiple Filters

You can specify any number of filters in any level of a query expression. Empty filters (`[]`) are not allowed.

A query that contains one or more filters returns the rightmost element that is not in a filter clause. For example:

```
book[excerpt]/author[degree]
```

The previous query returns author elements. It does not return degree elements. To be exact, this query returns all authors who have at least one degree if the author is of a book for which the document contains at least one excerpt. In other words, for all books in the current context that have excerpts, this query finds all authors with degrees.

The following query finds each book child of the current context that has an author with at least one degree:

```
book[author/degree]
```

The next query returns all books in the current context that have an excerpt and a title:

```
book[excerpt][title]
```

Filters and Attributes

Following is a query that finds all child elements of the current context with `specialty` attributes:

```
*[@specialty]
```

The following query returns all book children in the current context with `style` attributes:

```
book[@style]
```

The next query finds all book child elements in the current context in which the value of the `style` attribute of the book is equal to the value of the `specialty` attribute of the bookstore element:

```
book[/bookstore/@specialty = @style]
```

Wildcards in Queries

In a query, you can include an asterisk (*) to represent all elements. For example:

```
/bookstore/book/*
```

This query searches for all book elements in bookstore. For each book element, this query returns all child elements that the book element contains.

The `*` collection returns all elements that are children of the context node, regardless of their tag names.

The next query finds all `last-name` elements that are grandchildren of book elements in the current context:

```
book/*/last-name
```

The following query returns the grandchild elements of the current context.

```
*/*
```

Restrictions

Usually, the asterisk (*) returns only elements. It does not return processing instructions, attributes, or comments, nor does it include attributes or comments when it maintains a count of nodes. For example, the following query returns `title` elements. It does not return `style` attributes.

```
/bookstore/book/*[1]
```

Wildcards in strings are not allowed. For example, you cannot define a query such as the following:

```
/bookstore/book[author=" A* "]
```

Attributes

To use a wildcard for attributes, you can specify `@*`. For example:

```
/bookstore/book/@*
```

For each book element, this query returns all attributes. It does not return any elements.

Calling Functions in Queries

The XPath processor provides many functions that you can call in a query. This section provides some examples to give you a sense of how functions in queries work. Many subsequent sections provide information about invoking functions in queries. For a complete list of the functions you can call in a query, see [“XPath Functions Quick Reference”](#) on page 708.

Following is a query that returns a number that indicates how many book elements are in the document:

```
count(//book)
```

In format descriptions, a question mark that follows an argument indicates that the argument is optional. For example:

```
string substring(string, number, number?)
```

This function returns a string. The name of the function is `substring`. This function takes two required arguments (a string followed by a number) and one optional argument (a number).

Case Sensitivity and Blank Spaces in Queries

Queries are case sensitive. This applies to every part of the query, including operators, strings, element and attribute names, and function names.

For example, suppose you try this query:

```
/Bookstore
```

This query returns an empty set because the name of the document element is bookstore and not Bookstore.

Blank spaces in queries are not significant unless they appear within quotation marks.

Precedence of Query Operators

The precedence of query operators varies for XPath 1.0 and XPath 2.0, as shown in the following tables. In these tables, operators are listed in order of precedence, with highest precedence being first; operators in a given row have the same precedence.

Table 83. Query Operator Precedence – XPath 1.0

<i>Operation Type</i>	<i>XPath Operators</i>
Grouping	()
Filter	[]
Unary minus	-
Multiplication	*, div, mod
Addition	+, -
Relational (Comparison)	= != < <= > >=
Union	
Negation	not
Conjunction	and
Disjunction	or

Table 84. Query Operator Precedence – XPath 2.0

<i>Operation Type</i>	<i>XPath Operators</i>
Sequence separator	,
Conjunction	and
Type matching	instance of
Assertion	treat
Conversion test	castable
Conversion	cast

Table 84. Query Operator Precedence – XPath 2.0

<i>Operation Type</i>	<i>XPath Operators</i>
Relational (Comparison)	eg, ne, lt, le, gt, ge, =, !=, <, <=, >, >=, is, <<, >>
Range	to
Addition	+, -
Multiplication	*, div, idiv, mod
Unary	unary -, unary +
Union	union,
Select set	intersect, except
Navigation	/, //
Filter	[]

Specifying the Nodes to Evaluate

Consider the bookstore tree in the sample data. If you query the entire tree for all author elements, the result contains a number of author elements. If you query only one branch of the tree, the result contains only one author element. The result of the query depends on which nodes the XPath processor evaluates in the execution of the query.

This section discusses the following topics:

- [“Understanding XPath Processor Terms”](#) on page 657
- [“Starting at the Context Node”](#) on page 659
- [“About Root Nodes and Document Elements”](#) on page 659
- [“Starting at the Root Node”](#) on page 659
- [“Descending Along Branches”](#) on page 660
- [“Explicitly Specifying the Current Context”](#) on page 661
- [“Specifying Children or Descendants of Parent Nodes”](#) on page 662
- [“Examples of XPath Expression Results”](#) on page 662
- [“Syntax for Specifying an Axis in a Query”](#) on page 663

- [“Supported Axes”](#) on page 664
- [“Axes That Represent the Whole XML Document”](#) on page 669

Understanding XPath Processor Terms

To use the context operators, it is important to understand the following terms:

Axis

An *axis* specifies a list of nodes in relation to the context node. For example, the ancestor axis contains the ancestor nodes of the context node. The `child` axis contains the immediate children of the context node. See [“Syntax for Specifying an Axis in a Query”](#) on page 663.

Context Node

A *context node* is the node the XPath processor is currently looking at. The context node changes as the XPath processor evaluates a query. If you pass a document to the XPath processor, the root node is the initial context node. If you pass a node to the XPath processor, the node that you pass is the initial context node. During evaluation of a query, the initial context node is also the current node.

Context Node Set

A *context node set* is a set of nodes that the XPath processor evaluates.

Current Node

Current node is the node that the XPath processor is looking at when it begins evaluation of a query. In other words, the current node is the first context node that the XPath processor uses when it starts to execute the query. During evaluation of a query, the current node does not change. If you pass a document to the XPath processor, the root node is the current node. If you pass a node to the XPath processor, that node is the current node.

Document Element

The *document element* is the element in a document that contains all other elements. The document element is an immediate child of the root node. When you obtain the document element of a document, you obtain all marked-up text in that document.

Filter

A *filter* in a query specifies a restriction on the set of nodes to be returned. For example, the filter in the following query restricts the result set to book elements that contain at least one excerpt element:

```
book[excerpt]
```

Location Path Expression

A *location path expression* is an XPath expression. It has the following format:

```
[/]LocationStep[/LocationStep]...
```

Location Step

An XPath expression consists of one or more *location steps*. A location step has the following format:

```
[axis::]node_test[[filter] [filter]...]
```

Node Test

You apply a *node test* to a list of nodes. A node test returns nodes of a particular type or nodes with a particular name. For example, a node test might return all comment nodes, or all book elements.

Root Node

The *root node* is the root of the tree. It does not occur anywhere else in the tree. The document element node for a document is a child of the root node. The root node also has as children processing instructions and comment nodes representing processing instructions and comments that occur in the prolog and after the end of the document element.

Starting at the Context Node

Following is a query that looks for all child author elements in the current context:

```
author
```

This query is simply the name of the element you want to search for. If the context node is any one of the book elements, this query returns one author element. If the context node is any other node, this query returns the empty set.

About Root Nodes and Document Elements

A root node is the topmost node in the tree that represents the contents of an XML document. The root node can contain comments, a declaration, and processing instructions, as well as the *document element*. The document element is the element that contains all other elements; that is, the document element contains elements that are in the document but that are not immediate children of the root node.

Starting at the Root Node

To specify that the XPath processor should start at the root node when it evaluates nodes for a query, insert a forward slash (/) at the beginning of the query.

In an XML document, there is no text that corresponds to the root node. Externally, a root node is really a concept. Internally, there are data structures that represent this concept, but there is no text that you can point to and call a root node.

The following query instructs the XPath processor to start at the root node, as indicated by the forward slash at the beginning of the query.

```
/bookstore
```

This query searches the children of the root node for a bookstore element. Because the name of the document element is bookstore, the query returns it. If the name of the document element is not bookstore, this query returns an empty set.

The following query returns the entire document, starting with the root node. As you can see, the entire query is just a forward slash:

```
/
```

This query returns everything — comments, declarations, processing instructions, the document element, and any elements, attributes, comments, and processing instructions that the document element contains.

Descending Along Branches

Sometimes you want the XPath processor to evaluate all nodes that are descendants of a node and not just the immediate children of that node. This amounts to operating on a branch of the tree that forms the document.

To specify the evaluation of descendants that starts at the root node, insert two forward slashes (//) at the beginning of a query.

To specify the evaluation of descendants that starts at the context node, insert a dot and two forward slashes (./) at the beginning of the query.

Following is a query that finds all `last-name` elements anywhere in the current document:

```
//last-name
```

Suppose the context node is the first `book` element in the document. The following query returns a single `last-name` element because it starts its search in the current context:

```
./last-name
```

At the beginning of a query, `/` or `//` instructs the XPath processor to begin to evaluate nodes at the root node. However, between tag names, `/` is a separator, and `//` is an abbreviation for the `descendant-or-self` axis.

The `//` selects from all descendants of the context node set. For example:

```
book//award
```

This query searches the current context for `book` child elements that contain `award` elements. If the `bookstore` element is the context node, this query returns the two `award` elements that are in the document.

For the sample bookstore data, the following two queries are equivalent. Both return all `last-name` elements in the document.

```
//last-name  
//author//last-name
```

The first query returns all `last-name` elements in the sample document or in any XML document. The second query returns all `last-name` elements that are descendants of `author` elements. In the sample data, `last-name` elements are always descendants of `author` elements, so this query returns all `last-name` elements in the document. But in another XML document, there might be `last-name` elements that are not descendants of `author` elements. In that case, the query would not return those `last-name` elements.

Tip: `//` is useful when the exact structure is unknown. If you know the structure of your document, avoid the use of `//`. A query that contains `//` is slower than a query with an explicit path.

Explicitly Specifying the Current Context

If you want to explicitly specify the current context node, place a dot and a forward slash (`./`) in front of the query. This construct typically appears in queries that contain [filters](#) (see “[Filtering Results of Queries](#)” on page 649). The following two queries are equivalent:

```
./author  
author
```

Remember, if you specify the name of an element as a complete query (for example, `foo`), you obtain only the `foo` elements that are children of the current context node. You do not necessarily obtain all `foo` elements in the document.

You can also specify the dot notation (`.`) to indicate that you want the XPath processor to manipulate the current context. For example:

```
//title [.= "History of Trenton"]
```

In this example, the XPath processor finds all `title` elements. The dot indicates the context node. This causes the XPath processor to check each `title` in turn to determine whether its string value is `History of Trenton`.

Specifying Children or Descendants of Parent Nodes

Sometimes you want a query to return information about a sibling of the context node. One way to obtain a sibling is to define a query that navigates up to the parent and then down to the sibling.

For example, suppose the context node is the first author element. You want to find out the `title` associated with this author. The following query returns the associated `title` element:

```
../title
```

The double dot (`..`) at the beginning of the query instructs the XPath processor to select the parent of the context node. This query returns the `title` elements that are children of the first book element, which is the parent of the first author element. In the `bookstore.xml` document, there is only one such `title` element.

Now suppose that the context node is still the first author element and you want to obtain the `style` attribute for the book that contains this author. The following query does this:

```
../@style
```

The double dot notation need not appear at the beginning of a query. It can appear anywhere in a query string, just like the dot notation.

Examples of XPath Expression Results

Table 85 provides examples of XPath expression results:

Table 85. XPath Expression Results

<i>Expression</i>	<i>Result</i>
<code>/a</code>	Returns the document element of the document if it is an <code>a</code> element
<code>/a/b</code>	Returns all <code>b</code> elements that are immediate children of the document element, which is the <code>a</code> element
<code>//a</code>	Returns all <code>a</code> elements in the document
<code>//a/b</code>	Returns all <code>b</code> elements that are immediate children of <code>a</code> elements that are anywhere in the document

Table 85. XPath Expression Results

Expression	Result
a or ./a	Returns all a elements that are immediate children of the context node
a/b	Returns all b elements that are immediate children of a elements that are immediate children of the context node
a//b	Returns all b elements that descend from a elements that are immediate children of the context node
./a	Returns all a elements in the document tree branch that starts with the context node
../a	Returns all a elements in the document tree branch that are children of the parent node of the context node.

Syntax for Specifying an Axis in a Query

The previous sections provide examples of XPath expression syntax that uses abbreviations. This section introduces you to the axis syntax that many of the abbreviations represent. For a list of XPath abbreviations, see [“XPath Abbreviations Quick Reference”](#) on page 713.

You can use axis syntax to specify a location path in a query. An axis specifies the tree relationship between the nodes selected by an expression and the context node. The syntax for specifying an axis in a query is as follows:

```
axis_name::node_test
```

The axis names are defined in [“Supported Axes”](#) on page 664.

A node test is a simple expression that tests for a specified node type or node name. For example:

- `node()` matches any type of node.
- `text()` matches text or CDATA nodes.
- `comment()` matches comment nodes.
- `processing-instruction()` matches any processing instruction.
- `processing-instruction(name)` matches processing instructions whose target is *name*.

- *name* matches elements or attributes whose name is *name*.
- * matches any elements or any attributes.

XPath 2.0 adds additional tests, such as

- `element()` matches any element node
- `attribute()` matches any attribute node
- `document-node()` matches any document node

In addition, you can follow the node test with any number of filters.

Supported Axes

The XPath processor supports all XPath axes:

- `child`
- `descendant`
- `parent`
- `ancestor`
- `following-sibling`
- `preceding-sibling`
- `following`
- `preceding`
- `attribute`
- `namespace`
- `self`
- `descendant-or-self`
- `ancestor-or-self`

About the `child` Axis

The `child` axis contains the children of the context node. The following examples select the book children of the context node:

```
child::book  
book
```

If the context node is the bookstore element, each of these queries return the book elements in `bookstore.xml`. When you do not specify an axis, the `child` axis is assumed.

About the descendant Axis

The descendant axis contains the descendants of the context node. A *descendant* is a child or a child of a child, and so on. The descendant axis never contains attribute nodes. The following example selects the `first-name` element descendants of the context node:

```
descendant::first-name
```

If the context node is the `bookstore` element, this query returns all `first-name` elements in the document. If the context node is the first `publication` element, this query returns the `first-name` element that is in the `publication` element.

About the parent Axis

The parent axis contains the parent of the context node, if there is one. The following example selects the parent of the context node if it is a `title` element:

```
parent::title
```

If the first `title` element in `bookstore.xml` is the context node, this query returns the first `book` element.

Note that dot dot (`..`) is equivalent to `parent::node()`.

About the ancestor Axis

The ancestor axis contains the ancestors of the context node. The ancestors of the context node consist of the parent of the context node and the parent's parent, and so on. The ancestor axis always includes the root node, unless the context node is the root node. The following example selects the `book` ancestors of the context node:

```
ancestor::book
```

If the context node is the first `title` element in `bookstore.xml`, this query returns the first `book` element.

About the following-sibling Axis

The `following-sibling` axis contains all the siblings of the context node that come after the context node in document order. If the context node is an attribute node or namespace node, the `following-sibling` axis is empty. The following example selects the next book sibling of the context node:

```
following-sibling::book[position()=1]
```

If the context node is the first book element in `bookstore.xml`, this query returns the second book element.

About the preceding-sibling Axis

The `preceding-sibling` axis contains all the siblings of the context node that precede the context node in reverse document order. If the context node is an attribute node or namespace node, the `preceding-sibling` axis is empty. The following example selects the closest previous book sibling of the context node:

```
preceding-sibling::book[position()=1]
```

If the context node is the third book element in `bookstore.xml`, this query returns the second book element. If the context node is the first book element, this query returns the empty set.

About the following Axis

The `following` axis contains the nodes that follow the context node in document order. This can include

- Following siblings of the context node
- Descendants of following siblings of the context node
- Following siblings of ancestor nodes
- Descendants of following siblings of ancestor nodes

The `following` axis never includes

- Ancestors or descendants of the context node
- Attribute nodes
- Namespace nodes

The following example selects the book elements that are following siblings of the context node and that follow the context node in document order:

```
following::book
```

If the context node is the first book element, this query returns the last three book elements. If the context node is the second book element, this query returns only the third and fourth book elements.

About the preceding Axis

The preceding axis contains the nodes that precede the context node in reverse document order. This can include:

- Preceding siblings of the context node
- Descendants of preceding siblings of the context node
- Preceding siblings of ancestor nodes
- Descendants of preceding siblings of ancestor nodes

The preceding axis never includes

- Ancestors or descendants of the context node
- Attribute nodes
- Namespace nodes

The following example selects the book elements that are preceding siblings of the context node and that precede the context node in document order:

```
preceding::book
```

If the third book element is the context node, this query returns the first two book elements. If the first book element is the context node, this query returns the empty set.

About the attribute Axis

The attribute axis contains the attributes of the context node. The attribute axis is empty unless the context node is an element. The following examples are equivalent. They both select the style attributes of the context node. The at sign (@) is an abbreviation for the attribute axis.

```
attribute::style  
@style
```

If the context node is the second book element, this query returns a `style` attribute whose value is `textbook`.

About the namespace Axis

The namespace axis contains the namespace nodes that are in scope for the context node. This includes namespace declaration attributes for the

- Context node
- Ancestors of the context node

If more than one declaration defines the same prefix, the resulting node set includes only the definition that is closest to the context node.

If the context node is not an element, the namespace axis is empty.

For example, if an element is in the scope of three namespace declarations, its namespace axis contains three namespace declaration attributes.

About the self Axis

The `self` axis contains just the context node itself. The following example selects the context node if it is a `title` element:

```
self::title
```

Note that `dot` (`.`) is equivalent to `self::node()`.

About the descendant-or-self Axis

The `descendant-or-self` axis contains the context node and the descendants of the context node. The following example selects the `first-name` element descendants of the context node and the context node itself if it is a `first-name` element:

```
descendant-or-self::first-name
```

If the context node is the `first-name` element that is in the `author` element in the second book element, this query returns just the context node. If the context node is the second book element, this query returns the two `first-name` elements contained in the second book element.

Note that `//` is equivalent to `descendant-or-self::node()`, while `//name` is equivalent to `descendant-or-self::node()/child::name`.

About the ancestor-or-self Axis

The ancestor-or-self axis contains the context node and the ancestors of the context node. The ancestor-or-self axis always includes the root node. The following example selects the author element ancestors of the context node and the context node itself if it is an author element:

```
ancestor-or-self::author
```

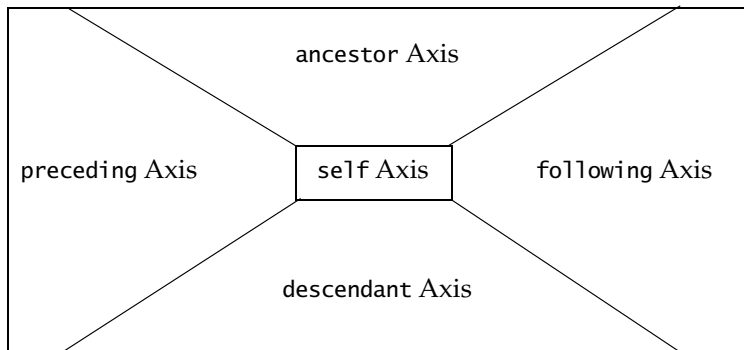
If the context node is the award element in the first book element, this query returns the first author element.

Axes That Represent the Whole XML Document

The following group of axes represent an entire XML document:

- ancestor
- preceding
- self
- following
- descendant

There is no overlap among these axes, as shown in the following figure:



Handling Strings and Text

This section includes the following topics:

- [“Searching for Strings”](#) on page 670
- [“Manipulating Strings”](#) on page 673
- [“Obtaining the Text Contained in a Node”](#) on page 676

Searching for Strings

This section provides information about searching for strings. This section discusses the following topics:

- [“Finding Identical Strings”](#) on page 670
- [“Finding Strings That Contain Strings You Specify”](#) on page 671
- [“Finding Substrings That Appear Before Strings You Specify”](#) on page 671
- [“Finding Substrings That Appear After Strings You Specify”](#) on page 672
- [“Finding Substrings by Position”](#) on page 672

Finding Identical Strings

In a document, you can search for text that is an exact match with what you specify in your query. For example, consider the following query:

```
//name [ . ="Lu" ]
```

This query finds all name elements that contain only the text Lu. It would return elements like these:

```
<name>Lu</name>  
<name>  
  <firstname>Lu</firstname>  
</name>
```

The same query does not return elements like these:

```
<name>Lu Chen</name>  
<name>  
  <firstname>Lu</firstname>  
  <lastname>Chen</lastname>  
</name>
```

The XPath processor does not return the first name element because the comparison is between "Lu" and "Lu Chen". The query does not return the second name element because the XPath processor concatenates the two strings "Lu" and "Chen" before it makes the evaluation. Consequently, the comparison is between "Lu" and "LuChen". Note that the XPath processor does not insert a space between text nodes that it concatenates.

Case Sensitivity

Searches are case sensitive. A search for "Lu" does not return "lu".

Finding Strings That Contain Strings You Specify

To obtain elements that contain a particular string, call the `contains()` function. The format is

```
boolean contains(string, string)
```

The `contains()` function returns `true` if the first argument string contains the second argument string, and otherwise returns `false`. For example, the following query returns all books that have a title that contains the string "Trenton":

```
/bookstore/book[contains(title, "Trenton")]
```

When the first argument is a node list, the XPath processor tests only the string value of the node in the node list that is first in document order. Any subsequent nodes are ignored.

Finding Substrings That Appear Before Strings You Specify

To obtain a substring that appears before a string you specify, call the `substring-before()` function. The format is

```
string substring-before(string, string)
```

The `substring-before()` function returns the substring of the first argument string that precedes the first occurrence of the second argument string in the first argument string. This function returns the empty string if the first argument string does not contain the second argument string. For example, the following call returns "1999":

```
substring-before("1999/04/01", "/")
```

Finding Substrings That Appear After Strings You Specify

To obtain a substring that appears after a string you specify, call the `substring-after()` function. The format is

```
string substring-after(string, string)
```

The `substring-after()` function returns the substring of the first argument string that follows the first occurrence of the second argument string in the first argument string. This function returns the empty string if the first argument string does not contain the second argument string. For example, the following call returns "04/01":

```
substring-after("1999/04/01", "/")
```

Finding Substrings by Position

To obtain a substring that is in a particular position within its string, call the `substring()` function. The format is

```
string substring(string, number, number?)
```

The `substring()` function returns the substring of the first argument, starting at the position specified in the second argument, with length specified in the third argument. For example, the following returns "234":

```
substring("12345", 2, 3)
```

If you do not specify the third argument, the `substring()` function returns the substring starting at the position specified in the second argument and continuing to the end of the string. For example, the following call returns "2345":

```
substring("12345", 2)
```

More precisely, each character in the string is considered to have a numeric position. The position of the first character is 1. The position of the second character is 2, and so on. The returned substring contains those characters for which the position of the character is greater than or equal to the rounded second argument and, if the third argument is specified, less than the sum of the value of the second and third arguments. The comparisons and addition used for the preceding follow the standard IEEE 754 rules. The

XPath processor rounds the second and third arguments as if by a call to the `round()` function. For example:

```
substring("12345", 1.5, 2.6) returns "234"  
substring("12345", 0, 3) returns "12"  
substring("12345", 0 div 0, 3) returns ""  
substring("12345", 1, 0 div 0) returns ""  
substring("12345", -42, 1 div 0) returns "12345"  
substring("12345", -1 div 0, 1 div 0) returns ""
```

Manipulating Strings

After you obtain a string, you might want to manipulate it and use the result in the query. This section describes functions that allow you to do this. It discusses the following topics:

- [“Concatenating Strings”](#) on page 673
- [“Determining the Number of Characters in a String”](#) on page 673
- [“Normalizing Strings”](#) on page 674
- [“Replacing Characters in Strings with Characters You Specify”](#) on page 674
- [“Converting Objects to Strings”](#) on page 675
- [“Finding Strings That Start with a Particular String”](#) on page 676

Concatenating Strings

To concatenate two or more strings, call the `concat()` function. The format is

```
string concat(string, string, {string}...)
```

The `concat()` function returns the concatenation of its arguments.

Determining the Number of Characters in a String

To obtain the number of characters in a string, call the `string-length()` function. The format is

```
number string-length(string?)
```

The `string-length()` function returns the number of characters in the string. If you omit the argument, it defaults to the string value of the context node.

Normalizing Strings

To strip leading and trailing white space from a string, call the `normalize-space()` function. The format is

```
string normalize-space(string?)
```

The `normalize-space()` function removes leading and trailing white space. White space consists of spaces, tabs, new lines, and returns.

If there are consecutive internal spaces, the `normalize-space()` function collapses the internal spaces into one space. The `normalize-space()` function returns the string with the extraneous white space removed. If you omit the argument, it defaults to the string value of the context node.

Replacing Characters in Strings with Characters You Specify

To replace characters in a string with other characters, call the `translate()` function. The format is

```
string translate(string, string, string)
```

The `translate()` function looks for characters in the first string that are also in the second string. For each such character, the `translate()` function replaces the character in the first string with a character from the third string. The replacement character is the character in the third string that is in the same position as the character in the second string that corresponds to the character being replaced. For example:

```
translate("bar", "abc", "ABC")
```

Execution of this function returns "BAR". Following is another example:

```
translate("---aaa---", "abc", "ABC")
```

Execution of this function returns "AAA". Sometimes there is a character in the second argument string with no character at a corresponding position in the third argument string. This happens when the second argument string is longer than the third argument string. In this case, the XPath processor removes occurrences of that character.

If a character occurs more than once in the second argument string, the first occurrence determines the replacement character. If the third argument string is longer than the second argument string, the XPath processor ignores the excess characters.

Converting Objects to Strings

In some situations, you might want to force a string comparison. The XPath processor performs a string comparison only when the operands are neither Boolean nor numeric values. If an operand is numeric or Boolean, call the `string()` function on it to convert it to a string. The format of the `string()` function is

```
string string(object?)
```

The `string()` function can convert any object to a string. If you omit the argument, it defaults to a node set with the context node as the only member. The string value of an element node is the concatenation of the string values of all text node descendants of the element node in document order.

Node Sets	When the <code>string()</code> function converts a node set to a string, it returns the string value of the node in the node set that is first in document order. If the node set is empty, the <code>string()</code> function returns an empty string.
Numbers	<p>The <code>string()</code> function converts numbers to strings as follows:</p> <ul style="list-style-type: none">● NaN (not a number) becomes "NaN"● Positive zero becomes "0"● Negative zero becomes "0"● Positive infinity becomes "Infinity"● Negative infinity becomes "-Infinity"● An integer becomes a sequence of digits with no leading zeros, for example, "1234". A negative integer is preceded by a minus sign, for example, "-1234".● A noninteger number becomes a sequence of digits with at least one digit before a decimal point and at least one digit after a decimal point, for example, "12.34". A negative noninteger number is preceded by a minus sign, for example, "-12.34". Leading zeros are not allowed unless there is only one to satisfy the requirement of a zero before the decimal point. Beyond the one required digit after the decimal point, there must be as many, but only as many, more digits as are needed to uniquely distinguish the number from all other IEEE 754 numeric values.
Boolean Values	The <code>string()</code> function converts the Boolean false value to the string "false", and the Boolean true value to the string "true".

Finding Strings That Start with a Particular String

To determine if a string starts with a particular string, specify the `starts-with()` function. The format is

```
boolean starts-with(string, string)
```

This function returns `true` if the first argument string starts with the second argument string, and otherwise returns `false`.

Obtaining the Text Contained in a Node

You can use the `string()` function to obtain the text in a node. The string value of an element node is the concatenation of the string values of all text node descendants of the element node in document order. Use one of the following formats:

```
string string(pathExpression)  
pathExpression
```

Replace *pathExpression* with the path of the node or nodes that contain the text you want. This can be a rooted path or a relative path. It need not be a single node. If you do not explicitly specify the `string()` function, you must specify *pathExpression* in a context where the XPath processor must treat it as a string, for example:

```
/bookstore/book[title = "Trenton Revisited"]
```

The XPath processor obtains the text contained in each `title` element and compares it with "Trenton Revisited". The XPath processor returns books with the title *Trenton Revisited*.

For additional information about the `string()` function, see [“Converting Objects to Strings”](#) on page 675.

Specifying Boolean Expressions and Functions

This section provides information on how to specify Boolean expressions and functions in queries. It includes the following topics:

- [“Using Boolean Expressions”](#) on page 677
- [“Calling Boolean Functions”](#) on page 678

Using Boolean Expressions

You can specify Boolean expressions in the subqueries in filters. You specify the Boolean AND, OR, and NOT operators like this:

- and
- or
- not

You can use parentheses to group collection specifications and operators for clarity or where the normal precedence is inadequate to express an operation.

Case Sensitivity

Operators are case sensitive. Spaces are not significant. You can omit them or include them for clarity.

Examples

The following query returns all authors who have at least one degree and one award:

```
author[degree and award]
```

The next query finds all authors who have at least one degree or award and at least one publication:

```
author[(degree or award) and publication]
```

Following is a query that finds all authors who have at least one degree and no publications:

```
author[degree and not(publication)]
```

Calling Boolean Functions

This section describes the Boolean functions that you can call in a query. The operations you can perform are

- [“Converting an Object to Boolean”](#) on page 678
- [“Obtaining Boolean Values”](#) on page 679
- [“Determining the Context Node Language”](#) on page 679

Converting an Object to Boolean

In some situations, you might want to force a Boolean comparison. The XPath processor performs a Boolean comparison if either operand is a Boolean value. Consequently, if neither operand is a Boolean value, call the `boolean()` function on one operand to convert it to a Boolean value. The XPath processor automatically converts the other operand to a Boolean value. The format of the `boolean()` function is
boolean `boolean(object)`

The `boolean()` function converts its argument to Boolean as follows:

- A number is `false` if and only if it is one of the following:
 - Positive zero
 - Negative zero
 - NaN (not a number)
- A node set is `false` if and only if it is empty.
- A string is `false` if and only if its length is 0.

The `boolean()` function is useful in comparisons. For example, the following query returns `b` elements that either contain both `c` and `d` elements as children or contain neither `c` nor `d` elements as children:

```
/a/b[boolean(c) = d]
```

This query is equivalent to the following query:

```
/a/b [(c and d) or (not(c) and not(d))]
```

Obtaining Boolean Values

To obtain the opposite Boolean value, call the `not()` function. The format is

```
boolean not(boolean)
```

The `not()` function returns `true` if its argument is `false`, and returns `false` if its argument is `true`. For example, the following query finds all authors who have publications but no degrees or awards:

```
author[not(degree or award) and publication]
```

To obtain the value `true`, call the `true()` function. The format is

```
boolean true()
```

The `true()` function returns `true`.

To obtain the value `false`, call the `false()` function. The format is

```
boolean false()
```

The `false()` function returns `false`.

Determining the Context Node Language

To determine whether the language of the context node is the language you expect it to be, call the `lang()` function. The format is

```
boolean lang(string)
```

The `lang()` function returns `true` or `false` depending on whether the language of the context node as specified by the `xml:lang` attribute is the same as, or is a sublanguage of, the language specified by the argument string. The language of the context node is determined by the value of the `xml:lang` attribute on the context node or, if the context node has no `xml:lang` attribute, by the value of the `xml:lang` attribute on the nearest ancestor of the context node that has an `xml:lang` attribute.

If there is no such attribute, then `lang()` returns `false`. If there is such an attribute, `lang()` returns `true` in the following situations:

- The attribute value is equal to the argument string.
- The attribute value has a suffix starting with a dash (-) such that the attribute value is equal to the argument string if you ignore the suffix.

In both situations, case is ignored. For example:

```
lang("en")
```

This returns `true` if the context node is any of these elements:

- `<para xml:lang="en"/>`
- `<div xml:lang="en"><para/></div>`
- `<para xml:lang="EN"/>`
- `<para xml:lang="en-us"/>`

Specifying Number Operations and Functions

This section includes the following topics:

- [“Performing Arithmetic Operations”](#) on page 680
- [“Calling Number Functions”](#) on page 681

Performing Arithmetic Operations

In queries, a number represents a floating-point number. A number can have any double-precision 64-bit format IEEE 754 value. This includes

- A special not-a-number (NaN) value
- Positive and negative infinity
- Positive and negative zero

The numeric operators convert their operands to numbers as if by calling the `number()` function. See [“Converting an Object to a Number”](#) on page 681.

You can use the following arithmetic operators in queries:

- `+` performs addition
- `-` performs subtraction

XML allows hyphens (-) in names. Consequently, the subtraction operator (-) typically needs to be preceded by white space. For example, `foo-bar` evaluates to a node set that contains the child elements named `foo-bar`. However, `foo - bar` evaluates to the difference between the result of converting the string value of the first `foo` child element to a number and the result of converting the string value of the first `bar` child to a number.

- `*` performs multiplication
- `mod` returns the remainder from a truncating division. For example:

- `5 mod 2` returns 1.
- `5 mod -2` returns 1.
- `-5 mod 2` returns -1.
- `-5 mod -2` returns -1.

The `mod` operator is the same as the `%` operator in Java and ECMAScript. But it does not perform the same operation as the IEEE remainder operation, which returns the remainder from a rounding division.

- `div` performs floating-point division according to IEEE 754.

Calling Number Functions

This section describes the number functions that you can call in a query. The operations you can perform are

- [Converting an Object to a Number](#) on page 681
- [Obtaining the Sum of the Values in a Node Set](#) on page 682
- [Obtaining the Largest, Smallest, or Closest Number](#) on page 682

Converting an Object to a Number

In some situations, you might want to force a numeric comparison. The XPath processor performs a numeric comparison if either operand is numeric and neither is Boolean. (If one operand is Boolean, the XPath processor converts the other to Boolean and performs a Boolean comparison.) However, if neither operand is a numeric or Boolean value, you can call the `number()` function on one operand to convert it to a numeric value. The XPath processor automatically converts the other operand to a numeric value.

To perform a numeric comparison, you must call the `number()` function to convert a Boolean operand, if there is one, to a numeric value.

The format of the `number()` function is

```
number number(object?)
```

If you omit the argument, the value of the argument defaults to a node set with the context node as its only member. [Table 86](#) shows how the `number()` function converts its argument to a number:

Table 86. `number()` Function Arguments and Converted Values

<i>Argument</i>	<i>Converted Value</i>
String that consists of optional white space followed by an optional minus sign followed by a number followed by white space	IEEE 754 number that is nearest to the mathematical value represented by the string.
Any other string	NaN (not a number)
Boolean true	1
Boolean false	0
Node set	First the XPath processor converts the node set to a concatenated string as if by a call to the <code>string()</code> function for the first node in the node set, in document order. The XPath processor then converts this string the same way as it would a string argument.

Obtaining the Sum of the Values in a Node Set

To obtain the sum of the values of the nodes in a set, call the `sum()` function. The format is

```
number sum(node-set)
```

For each node in the argument *node-set*, the XPath processor converts the string value of the node to a number. The `sum()` function returns the sum of these numbers.

Obtaining the Largest, Smallest, or Closest Number

To obtain the largest integer that is not greater than a particular number, call the `floor()` function. The format is

```
number floor(number)
```

The `floor()` function returns the largest (closest to positive infinity) number that is not greater than the argument and that is an integer. For example:

- `floor(5.3)` returns 5
- `floor(-5.3)` returns -6

To obtain the smallest integer that is not less than a particular number, call the `ceiling()` function. The format is

```
number ceiling(number)
```

The `ceiling()` function returns the smallest (closest to negative infinity) number that is not less than the argument and that is an integer. For example:

- `ceiling(5.3)` returns 6
- `ceiling(-5.3)` returns -5

To obtain the closest integer to a particular number, call the `round()` function. The format is

```
number round(number)
```

The `round()` function returns the number that is closest to the argument and that is an integer. If there are two such numbers, the function returns the one that is closest to positive infinity. For example:

- `round(5.3)` returns 5
- `round(5.6)` returns 6
- `round(5.5)` returns 6

Comparing Values

In queries, you can specify operators that compare values. Comparison operations return Boolean values. If you want to obtain the nodes for which a comparison tests true, enclose the comparison in a filter.

This section discusses the following topics:

- [“About Comparison Operators”](#) on page 684
- [“How the XPath Processor Evaluates Comparisons”](#) on page 684
- [“Comparing Node Sets”](#) on page 685
- [“Comparing Single Values With = and !=”](#) on page 686
- [“Comparing Single Values With <=, <, >, and >=”](#) on page 687
- [“Priority of Object Types in Comparisons”](#) on page 687

- [“Examples of Comparisons”](#) on page 688
- [“Operating on Boolean Values”](#) on page 688

About Comparison Operators

The comparison operators you can specify are listed in [Table 87](#):

Table 87. Comparison Operator Descriptions

<i>Operator</i>	<i>Description</i>
=	Equality
!=	Inequality
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal

You can specify single or double quotation marks (' or ") as string delimiters in expressions. This makes it easier to construct and pass queries from within scripting languages.

How the XPath Processor Evaluates Comparisons

A query can compare values of elements. For example:

```
last-name [. = "foo"]
```

The XPath processor compares the value of each `last-name` element in the current context with the value `"foo"`. The result of each comparison is a Boolean value. The XPath processor returns the `last-name` elements for which the comparison yields `true`.

As mentioned before in [“Filtering Results of Queries”](#) on page 649, the XPath processor evaluates filters with respect to a context. For example, the expression `book[author]` means for every `book` element that is found, determine whether it has an `author` child element. Likewise, `book[author = "Bob"]` means for every `book` element that is found, determine whether it contains an `author` child element whose value is `"Bob"`.

Comparisons are case sensitive. For example, "Bob" and "bob" are not considered to be equal.

Remember that comparisons return Boolean values. For example:

```
/bookstore/book/author[last-name="Bob"]
```

You might think that this query returns authors whose last name is "Bob". But this is not the case. This query returns a single Boolean value. It tests each `last-name` element to determine if its value is "Bob". As soon as the XPath processor finds a `last-name` element that tests true, it returns `true`. If no nodes test true, this query returns `false`.

To obtain author elements whose last name is "Bob", enclose the comparison in a filter as follows:

```
/bookstore/book/author[last-name="Bob"]
```

Comparing Node Sets

You can compare

- Two node sets
- A node set and a number
- A node set and a string
- A node set and a Boolean value

Two Node Sets

Suppose the objects you want to compare are both node sets. The result is `true` only in the following case. There is a node in the first node set and a node in the second node set such that the result of performing a comparison on the values of the two nodes is `true`. For string values, the comparison can be `=` or `!=`. For numeric values, the comparison can also be `<`, `>`, `<=`, or `>=`.

A Node Set and a Number

Now suppose one object to be compared is a node set and the other is a number. The XPath processor searches for a node in the node set that yields a true result when its number value is compared with the number that is not in the node set. If necessary, the XPath processor uses the `number()` function to convert values to numeric values. If and only if the XPath processor finds such a node, the result is `true`.

A Node Set and a String

Sometimes you want to compare a node set with a string. The XPath processor searches for a node in the node set that yields a true result when its string value is compared with the string that is not in the node set. If necessary, the XPath processor uses the `string()` function to convert values to string values. If and only if the XPath processor finds such a node, the result is `true`.

A Node Set and a Boolean Value

Finally, suppose you want to compare a node set with a Boolean value. This tests true if and only if the result of performing the comparison on the Boolean value and on the result of converting the node set to a Boolean value using the `boolean()` function is `true`.

Comparing Single Values With = and !=

When neither object to be compared is a node set and the operator is `=` or `!=`, the XPath processor compares the objects by converting them to a common type and then comparing them. The XPath processor converts the objects to a common type as follows:

- If at least one object to be compared is Boolean, the XPath processor converts the other object to Boolean as if by applying the `boolean()` function.
- If at least one object to be compared is a number, and neither is Boolean, the XPath processor converts the nonnumber object to a number as if by applying the `number()` function.

If the objects to be compared are neither Boolean nor numeric, the XPath processor compares the string values of the objects as if by applying the `string()` function to each object.

The `=` comparison returns `true` if and only if the objects are equal. The `!=` comparison returns `true` if and only if the objects are not equal. Numbers are compared for equality according to IEEE 754. Two Boolean values are equal if either both are `true` or both are `false`. Two strings are equal if and only if they both consist of the same sequence of Universal Character Set (UCS) characters.

Note Use single or double quotes to specify string values being used with a comparison operator.

Comparing Single Values With `<=`, `<`, `>`, and `>=`

When neither object to be compared is a node set and the operator is `<=`, `<`, `>=`, or `>`, the XPath processor performs the comparison by converting both objects to numbers and comparing the numbers according to IEEE 754.

Table 88. Comparison Operator Descriptions

<i>Comparison</i>	<i>True If and Only If</i>
<code><</code>	The first number is less than the second number.
<code><=</code>	The first number is less than or equal to the second number.
<code>></code>	The first number is greater than the second number.
<code>>=</code>	The first number is greater than or equal to the second number.

The XPath processor always evaluates these comparisons in terms of numbers. You cannot use the less than and greater than operators to order strings. This is especially important to remember when you compare a number with a string. For example, suppose you want to evaluate the expression

```
a < "foo"
```

The return value is always `false`. This is because `number("foo")` returns `NaN`, and the resulting comparison, shown below, is always `false`.

```
a < NaN
```

Priority of Object Types in Comparisons

When the XPath processor performs a comparison, if either operand is a Boolean value, the XPath processor automatically converts the other operand to a Boolean value, if necessary, and makes a Boolean comparison.

If either operand is numeric and neither operand is Boolean, the XPath processor automatically converts the other operand to a numeric value, if necessary, and performs a numeric comparison.

If neither operand is numeric or Boolean, the XPath processor performs a string comparison.

Examples of Comparisons

The following query finds all authors whose last name is Bob:

```
author[last-name = "Bob"]
```

The next query finds authors whose degrees are not from Harvard:

```
author/degree[@from != "Harvard"]
```

The following query returns prices that are greater than 20 dollars. This assumes that the current context contains one or more price elements.

```
price [. > 20]
```

Operating on Boolean Values

You can use the = or != operator to compare Boolean values. If you try to use any other operator to compare Boolean values, you receive an error.

Finding a Particular Node

To find a specific node within a set of nodes, enclose an integer within brackets ([]). The integer indicates the position of the node relative to its parent. This section discusses the following topics:

- [“About Node Positions”](#) on page 689
- [“Determining the Position Number of a Node”](#) on page 689
- [“Positions in Relation to Parent Nodes”](#) on page 690
- [“Finding Nodes Relative to the Last Node in a Set”](#) on page 691
- [“Finding Multiple Nodes”](#) on page 691
- [“Examples of Specifying Positions”](#) on page 692
- [“Finding the First Node That Meets a Condition”](#) on page 692
- [“Finding an Element with a Particular ID”](#) on page 692
- [“Obtaining Particular Types of Nodes By Using Node Tests”](#) on page 694

See also [“Obtaining the Current Node for the Current XSLT Template”](#) on page 702.

About Node Positions

The node positions for a node set start with 1. Evaluation of the position number is always based on document order. For example, the following query returns the first author element in the current context:

```
author[1]
```

The next query finds the author elements (in the current context) that contain a first-name element. The query returns the third such author element.

```
(author[first-name])[position()=3]
```

When you specify an integer in brackets, it is equivalent to calling the `position()` function. For example, the following queries both return the third `y` child element of each `x` child element in the current context:

```
x/y[3]  
x/y[position()=3]
```

Tip: If you do not know the position of the node you want, a call to the `position()` function might help you. See [“Determining the Position Number of a Node”](#) on page 689.

The return value of the `position()` function depends on the specified axis. For example, suppose the axis is one of the reverse axes, such as `preceding`, `ancestor`, or `preceding-sibling`. The `position()` function returns the n th one in reverse document order that falls in the specified axis.

Determining the Position Number of a Node

The `position()` function returns an integer that indicates the position of the node within the parent. Positions start with 1; a node with a position of 1 is always the first node in the collection.

For example, the following query returns the first three degree elements in the document:

```
(//degree)[position() < 4]
```

The next query finds the first two book children in the current context:

```
book[position() <= 2]
```

The XPath processor executes the `position()` function with respect to the parent node. Consider the following data:

```
<x>
  <y/>
  <y/>
</x>
<x>
  <y/>
  <y/>
</x>
```

The following expression returns the first `y` element contained in each `x` element:

```
x/y[position() = 1]
```

For more information, see also [“Finding an Element with a Particular ID”](#) on page 692.

Positions in Relation to Parent Nodes

Positions are relative to the parent. Consider the following data, which has line numbers on the left for explanation only.

```
1 <x>
2   <z>
3   <z/>
4 </x>
5 <x>
6   <y>
7   <y/>
8 </x>
9 <x>
10  <y>
11  <y/>
12 </x>
```

The following query returns the first `y` element contained in each `x` element. It returns the elements on lines 6 and 10. The XPath processor finds all `x` elements. For each `x` element, the XPath processor then returns the first `y` element it finds.

```
x/y[1]
```

The next query returns the first `y` element that is contained in an `x` element that is in the context node set. It returns the element on line 6. The XPath processor finds all `y` elements inside `x` elements. The XPath processor then returns the first element in that set.

```
(x/y)[1]
```


The next query returns the empty set. The XPath processor finds the first x element. It then searches that first x element for the first y . Because the first x element does not contain a y element, this query returns the empty set.

```
x[1]/y[1]
```

Finding Nodes Relative to the Last Node in a Set

To obtain nodes relative to the last node in the set, use the `position()` and `last()` functions with arithmetic. For example, the following queries both obtain the last author element in the current context:

```
author [position() = last()]
author [last()]
```

The following queries both return the next-to-last author element:

```
author [position() = last() - 1]
author [last() - 1]
```

For information about `position()`, see [“Determining the Position Number of a Node”](#) on page 689. For information about `last()`, see [“Determining the Context Size”](#) on page 700.

Finding Multiple Nodes

To obtain several nodes in one operation, use the `and` or the `or` operator with the `position()` and `last()` functions. For example, the following query returns the first and the last author nodes in the current context:

```
author [(position() = 1) or (position() = last())]
```

You can also specify a range of nodes. For example, the next query returns the second, third, and fourth author elements:

```
author [(position() >= 2) and ( position() <= 4)]
```

To obtain a range of nodes, m to n , relative to the last node, use the following format:

```
( $m$  <= position()) and (position() <=  $n$ )
```

For example, the following query obtains the last five nodes in the current context:

```
author [(last() - 4) <= position() and (position() <= last())]
```

Examples of Specifying Positions

The following query finds the first and fourth author elements:

```
author [(position() = 4) or (position() = 1)]
```

The next query finds the first, the third through the fifth, and the last author elements:

```
author [(position() = 1) or  
        (position() >= 3 and position() <= 5) or  
        (position() = last())]
```

The XPath processor removes duplicate values. For the previous query, if there are only five elements in the collection, the query returns only one copy of the fifth element.

The next example finds all author elements in which the first degree is a Ph.D.:

```
author[degree[1] = "Ph.D."]
```

Finding the First Node That Meets a Condition

Suppose you want to obtain from a collection the first node that meets a certain condition. For example, you want the first book whose author's last name is Bob. You can specify the following query:

```
(//book[author/last-name="Bob"])[position()=1]
```

When the XPath processor evaluates this expression, it creates a collection of book elements where the author's last name is Bob. The XPath processor then returns the first node in this collection.

The following two expressions appear to also return the first book whose author's last name is Bob, but they do not. Instead, these queries both return a book whose author's last name is Bob only if that book is the first book in the document.

```
//book[author/first-name="Bob"][position()=1]  
//book[author/first-name="Bob" and position() = 1]
```

Finding an Element with a Particular ID

To obtain the element that has a particular identifier (ID), the DTD must specify an attribute for that element. The type of this attribute must be ID. The name of the attribute

is not significant, though it is typically `id`. If there is such an attribute, you can call the `id()` function to obtain the element with a particular ID. The format is

```
node-set id(object)
```

The `id()` function evaluates to a set. It ignores the context node set except to evaluate the function's argument. The result set contains an element node that has an attribute of type ID whose value is identical to the string the *object* argument evaluates to. The element node can appear anywhere in the document that is being queried.

For example:

```
id("special")
```

This query searches for an element that has an attribute whose

- Type is ID
- Value is `special`

Details about working with IDs are in the following topics:

- [“The `id\(\)` Function's Argument”](#) on page 693
- [“Unique IDs”](#) on page 693

The `id()` Function's Argument

When the `id()` function's argument is of type `node-set`, the result is the union of the results of applying `id()` to the string value of each of the nodes in the argument node set.

When the argument of `id()` is any other type, the XPath processor converts the argument to a string as if by a call to the `string()` function. The XPath processor splits the string into a white-space-separated list of tokens. The result is a node set that contains the elements in the same document as the context node that have a unique ID equal to any of the tokens in the list.

Unique IDs

An element node can have a unique ID. This is the value of the attribute that is declared in the DTD as type ID. No two elements in a document can have the same unique ID. If an XML processor reports two elements in a document as having the same unique ID (which is possible only if the document is invalid), the second element is treated as not having a unique ID.

If a document does not have a DTD, the `id()` function always returns an empty node list.

Obtaining Particular Types of Nodes By Using Node Tests

The node tests allow you to obtain nodes according to their type. *Node test* is an XPath term. Although a node test looks like a function, it is not a function.

You can use node tests with filters and position specifiers. They resolve to the set of children of the context node that meets the restrictions you specify.

Node tests for XPath 2.0 add to the set of node tests supported for XPath 1.0. Node tests common to both XPath 1.0 and XPath 2.0 are shown in [Table 89](#). Node tests unique to XPath 2.0 are shown in [Table 89](#).

Table 89. Node Test Return Values Common to XPath 1.0 and XPath 2.0

Node Test	Node Type Returned
<code>comment()</code>	Comment nodes.
<code>node()</code>	All nonattribute nodes.
<code>processing-instruction("name")</code>	Processing instruction nodes. The <code>processing-instruction()</code> node test selects all processing instructions. When this node test specifies a literal argument, it selects any processing instruction that has a name equal to the value of the argument. If there is no argument, this node test selects all processing instructions.
<code>text()</code>	Text nodes and CDATA nodes.

Table 90. Node Test Return Values Unique to XPath 2.0

Node Test	Node Type Returned
<code>attribute()</code>	Matches any attribute node.
<code>document-node()</code>	Matches any document node.
<code>element()</code>	Matches any element node.
<code>item()</code>	Matches any single item.

For each `p` element in the current context, the following query returns its second `text` child node:

```
p/text()[2]
```

Following is a query that finds the third comment child in each foo element anywhere in the document:

```
//foo/comment()[3]
```

About the Document Object

In the Document Object Model (DOM), a document contains comments, processing instructions, and declarations, as well as the document element. As in XPath, the root node is the root of the DOM tree, and the root node is not the same as the document element. This allows comments, declarations, and processing instructions at the document entity level.

For example, the following query finds all comments at the document entity level. In other words, it finds all comments that are immediate children of the root node.

```
/comment()
```

This query returns the comment at the beginning of the bookstore.xml file:

```
"This file represents a fragment of a book store inventory database."
```

Getting Nodes of a Particular Type

A query like the following returns all the comments in a document:

```
//comment()
```

The following query returns the third comment in the document.

```
(//comment())[3]
```

Obtaining a Union

Specify the | operator to combine collection sets. For example, the following query returns all last-name elements and all first-name elements in the current context:

```
first-name | last-name
```

The result set can contain zero or more of each element that the | operator applies to. For example, using the previous query, it is possible for the query to contain only first-name

Writing XPath Expressions

elements if no last-name elements are found. The following query finds all book elements and magazine elements in the bookstore element:

```
/bookstore/book | /bookstore/magazine
```

The next query finds all books and all authors in the current context:

```
book | book/author
```

The next query returns the first names, last names, and degrees of authors of books or magazines in the current context:

```
((book | magazine)/author/first-name) |  
(book | magazine)/author/last-name |  
(book | magazine)/author/degree )
```

A union can appear only as the first step in a location path expression. Consequently, the following is incorrect because there is a union specification that is not in the first step of a location path expression.

```
(book | magazine)/author/(first-name | last-name | degree)
```

The following query finds all books for which the author's first name is Bob and all magazines with prices less than 10 dollars:

```
/bookstore/book[author/first-name = "Bob"] | magazine[price < 10]
```

Obtaining Information About a Node or a Node Set

In a query, you can perform the following operations to obtain information about a node:

- [“Obtaining the Name of a Node”](#) on page 697
- [“Obtaining Namespace Information”](#) on page 697
- [“Obtaining the URI for an Unparsed Entity”](#) on page 700
- [“Determining the Number of Nodes in a Collection”](#) on page 700
- [“Determining the Context Size”](#) on page 700

Obtaining the Name of a Node

The `name()` function returns a string that contains the tag name of the node, including the namespace prefix, if any.

The following query returns the name of the third element in bookstore, which is "magazine".

```
name(/bookstore/*[3])
```

Wildcards

An asterisk (`*`) specifies a wildcard name for element names. If there are comments before the third element in the preceding example, this query does not include them in the count. See [“Filtering Results of Queries”](#) on page 649.

Note Remember, an asterisk that is not preceded by an at sign (`@`) never returns attributes. The XPath processor does not include attributes in node counts. See [“Attributes and Wildcards”](#) on page 649.

Obtaining Namespace Information

You can call functions to obtain namespace information. This topic discusses

- [“Obtaining the Namespace URI”](#) on page 698
- [“Obtaining the Local Name”](#) on page 698
- [“Obtaining the Expanded Name”](#) on page 698

In addition to a discussion of the functions you call, this section covers the following:

- [“Specifying Wildcards with Namespaces”](#) on page 699
- [“Examples of Namespaces in Queries”](#) on page 699

Obtaining the Namespace URI

To obtain the URI for a namespace, call the `namespace-uri()` function. The format is

```
string namespace-uri(node-set?)
```

The `namespace-uri()` function returns the namespace URI of the expanded name of the node in the *node-set* argument that is first in document order. If the *node-set* argument is empty, the first node has no expanded name, or the namespace URI of the expanded name is null, the XPath processor returns an empty string. If you omit the argument, it defaults to a node set with the context node as its only member.

Call the `namespace-uri()` function on element or attribute nodes. For example, the query

```
/bookstore/my:book/namespace-uri()
```

returns the string

```
"http://www.placeholder-name-here.com/schema/"
```

For any other type of node, the XPath processor always returns an empty string.

Obtaining the Local Name

To obtain the local portion of a node name, excluding the prefix, call the `local-name()` function. The format is

```
string local-name(node-set?)
```

The `local-name()` function returns the local part of the expanded name of the node in the *node-set* argument that is first in document order. If the *node-set* argument is empty or the first node has no expanded name, the function returns an empty string. If you omit the argument, it defaults to a node set with the context node as its only member. For example, the following query returns `my:book` nodes:

```
/bookstore/my:*[local-name()='book']
```

Obtaining the Expanded Name

To obtain the expanded name of a node, call the `name()` function. The expanded name is the namespace prefix, if any, plus the local name. The format is

```
string name(node-set?)
```


The `name()` function returns a string that represents the expanded name of the node in the *node-set* argument that is first in document order. The returned string represents the expanded name with respect to the namespace declarations in effect on the node whose expanded name is being represented.

Typically, this is the name in the XML source. This need not be the case if there are namespace declarations in effect on the node that associate multiple prefixes with the same namespace.

If the *node-set* argument is empty or the first node has no expanded name, the XPath processor returns an empty string. If you omit the argument, it defaults to a node set with the context node as its only member.

Except for element and attribute nodes, the string that the `name()` function returns is the same as the string the `local-name()` function returns.

Specifying Wildcards with Namespaces

Element and attribute names that include colons (:) can include wildcards; that is, asterisks (*). For example, queries can include `*:*`, `*:a`, or `a:*`.

Examples of Namespaces in Queries

The following example finds all book elements in the current context. This query does not return any book elements that are not in the default namespace. For example, it does not return `my:book` elements.

```
book
```

The next query finds all book elements with the prefix `my`. This query does not return unqualified book elements; that is, book elements in the default namespace.

```
my:book
```

The following query finds all book elements with a `my` prefix that have an author subelement:

```
my:book[author]
```

The following query finds all book elements with a `my` prefix that have an author subelement with a `my` prefix:

```
my:book[my:author]
```

The next example returns the `style` attribute with a `my` prefix for book elements in the current context:

```
book/@my:style
```

Obtaining the URI for an Unparsed Entity

To obtain the URI for an unparsed entity, call the `unparsed-entity-uri()` function. The format is

```
stringunparsed-entity-uri(string)
```

This function returns the URI of the unparsed entity with the specified name that is in the same document as the context node. If there is no such entity, this function returns an empty string.

Determining the Number of Nodes in a Collection

To obtain the number of nodes in a node set, call the `count()` function. The format is

```
number count(node-set)
```

The `count()` function returns the number of nodes in the specified set. For example, the following query finds all authors who have more than ten publications:

```
//author[count(publications) > 10]
```

To obtain the number of nodes in the current context, call the `last()` function, described in the next section.

Determining the Context Size

To obtain the number of nodes in the current context, call the `last()` function. The format is

```
number last()
```

The `last()` function returns a number equal to the context size of the expression evaluation context. Essentially, the `last()` function returns the position number of the last node in document order for the current context. For example, the following query returns

all books if there are three or more of them. There are three book elements in the current context. Consequently, this query returns three book elements.

```
/bookstore/book[last() >= 3]
```

Using XPath Expressions in Stylesheets

This section provides information about using XPath expressions in stylesheets. It includes the following topics:

- [“Using Variables”](#) on page 701
- [“Obtaining System Properties”](#) on page 701
- [“Determining If Functions Are Available”](#) on page 702
- [“Obtaining the Current Node for the Current XSLT Template”](#) on page 702
- [“Finding an Element with a Particular Key”](#) on page 703
- [“Generating Temporary IDs for Nodes”](#) on page 705

Using Variables

In a query that you specify in a stylesheet, you can refer to variables that you defined elsewhere in the stylesheet. Use the following format to refer to a variable:

```
$variable_name
```

In a stylesheet, you can define variables with either of the following instructions:

- `xs1:param` on page 436
- `xs1:variable` on page 446

Obtaining System Properties

In a query in a stylesheet, there are three system properties for which you can obtain information:

- `xs1:version` returns 1.0 as the version of XSLT that the Stylus Studio XSLT processor implements.
- `xs1:vendor` returns DataDirect as the vendor of Stylus Studio’s XSLT processor.
- `xs1:vendor-url` returns `http://www.stylusstudio.com` as the vendor URL.

To obtain this information, call the `system-property()` function. The format is

```
object system-property(string)
```

The string you specify must identify one of the three properties and must be a qualified name. This function returns an object that represents the value of the system property you specify.

Determining If Functions Are Available

In a query in a stylesheet, to determine whether the XPath processor supports a particular function, call the `function-available()` function. The format is

```
boolean function-available(string)
```

Specify a string that identifies the name of the function. The XPath processor returns `true` if it implements that function.

Obtaining the Current Node for the Current XSLT Template

In a stylesheet, the current node is the node for which the XSLT processor instantiates a template. When the XPath processor evaluates an expression during stylesheet processing, the initial context node for the expression is set to the current node for the stylesheet instruction that contains the expression. Because the context node can change during evaluation of subexpressions, it is useful to be able to retrieve, from within a subexpression, the original context node for which the expression is being evaluated. You can use the `current()` function for this purpose. The format is

```
node-set current()
```

The `current()` function returns a node set that contains only the current node for the current template. The `current()` function is specified in the W3C XSLT Recommendation.

For example, the following stylesheet causes the XSLT processor to pass the bookstore node to the outer `xs1:for-each` instruction:

```
<xs1:stylesheet
  xmlns:xs1="http://www.w3.org/XSL/Transform" version="1.0" >
  <xs1:template match="/">
    <xs1:for-each select="bookstore">
      <xs1:for-each select=
        "book[@style=current()/@specialty]">
        ...
      </xs1:for-each>
    </xs1:for-each>
  </xs1:template>
</xs1:stylesheet>
```

The bookstore node is the current node within the outer `xs1:for-each` instruction. Within the inner `xs1:for-each` instruction, a book node is the current node.

The `current()` function in the inner expression returns the bookstore element because the bookstore element is the current node for the inner `xs1:for-each` instruction. The result of the query contains book elements if the value of their `style` attribute is the same as the value of the `specialty` attribute of the bookstore element (`novel`).

Suppose the `select` attribute in the inner `xs1:for-each` instruction specified the dot (`.`) instead of the `current()` function:

```
<xs1:for-each select="book[@style=../@specialty]">
```

In a query, the dot specifies the context node. This query would return a book if the value of its `style` attribute was the same as the value of its `specialty` attribute.

You can nest `xs1:for-each` instructions more than one level deep. In any given nested `xs1:for-each` instruction, the `current()` function returns the current node for the closest enclosing `xs1:for-each` instruction.

Finding an Element with a Particular Key

The `key()` function, defined in the XSLT Recommendation, obtains the node whose key value matches the specified key. The format is

```
node-set key(string, object)
```

The first argument specifies the name of the key. The value of this argument must be a qualified name. The second argument specifies the node or nodes to examine. When the

second argument is a node set, the result is the union of the results of applying the `key()` function to the string value of each of the nodes in the set. When the second argument is any other type, the XPath processor converts the argument to a string, as if by a call to the `string()` function. The `key()` function returns a node set that contains the nodes in the same document as the context node that have a value for the named key that is equal to this string.

For example, the `videos.xml` document, which is in the `examples` directory of the Stylus Studio installation directory, contains the following elements:

```
<result>
  <actors>
    <actor id="00000003">Jones, Tommy Lee</actor>
    ...
  </actors>
  <videos>
    <video id="id1235AA0">
      <title>The Fugitive</title>
      ...
      <actorRef>00000003</actorRef>
      <actorRef>00000006</actorRef>
      ...
    </video>
    ...
  </videos>
</result>
```

When you display information about a video in a Web browser, you want to display the names of the actors. Because the actors are referenced only by an ID number, you create a key table in your stylesheet:

```
<xsl:key
  name="actors"
  match="/result/actors/actor"
  use="@id"/>
```

This indexes all actors by their ID. To process a video, your stylesheet specifies the following:

```
<xsl:for-each select="actorRef">
  <xsl:value-of select="key('actors', .)"/>
</xsl:for-each>
```

This instructs the XPath processor to look up the actor element in the actors key table by using the actorRef element as a key.

Generating Temporary IDs for Nodes

The `generate-id()` function, defined in the XSLT recommendation, generates temporary IDs for nodes.

Caution The ID generated by the `generate-id()` function is not an object ID. The value generated by the `generate-id()` function is guaranteed to be the same only during an XSL transformation. If the source document changes, the value for this ID can change.

Format

The format for the `generate-id()` function is as follows:

```
string generate-id(node-set?)
```

The `generate-id()` function returns a string that uniquely identifies the node in *node-set* that is first in document order. This string starts with `x1n` and ends with eight hexadecimal digits. Syntactically, the string is an XML name.

If the *node-set* argument is empty, the `generate-id()` function returns an empty string. If you omit the *node-set* argument, the `generate-id()` function generates a temporary ID for the context node.

Accessing Other Documents During Query Execution

During execution of a query, you might want to access data in another document. To do this, call the `document()` function.

This section discusses the following topics:

- [“Format of the document\(\) Function”](#) on page 706
- [“When the First Argument is a Node Set”](#) on page 706
- [“Specification of Second Argument”](#) on page 706
- [“Example of Calling the document\(\) Function”](#) on page 707

Format of the document() Function

To query multiple documents with a single query, call the `document()` function in a query. During execution of a query on a particular document, this function allows you to access another XML document.

The format for the `document()` function is

```
node-set document(object, node-set?)
```

The XPath processor examines the first argument. If it is a single value (that is, it is not a node set) the XPath processor converts it to a string, if it is not already a string. Separate directory names and the file name with a forward slash (/). See the following format:

This string must be an absolute path. The XPath processor retrieves the specified document. The new context node is the root node of this document. Suppose you invoke the `document()` function and the requested document does not exist. If the invocation is in a stylesheet, the XPath processor returns an empty node set. If the invocation is anywhere else, the XPath processor returns an error message.

When the First Argument is a Node Set

It is possible for the first argument of the `document()` function to be a node set. In this case, the result is as if you had called the `document()` function on each node in this node set. That is, the first argument of the `document()` function is each node in the node set in turn. The second argument, if there is one, is the same for each iteration of the `document()` function. This allows you to obtain the contents of multiple documents.

Specification of Second Argument

If you specify a second argument, it must be a node set. The XPath processor examines the first node (in the context of document order) in the node set to determine the document that this node belongs to. The XPath processor retrieves the name of the directory that contains this document and appends the relative path from the first argument to the name of the directory. This creates an absolute path, and the XPath processor retrieves the specified document.

If there is no second argument, the query must be an expression in an XSLT stylesheet. The XPath processor appends the relative path to the name of the directory that contains the XSLT stylesheet. This allows the query to examine the stylesheet itself.

Example of Calling the document() Function

Suppose you have the following XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <bookstore>bookstore1:bookstore1.xml</bookstore>
  <bookstore>bookstore2:bookstore2.xml</bookstore>
  <bookstore>bookstore3:bookstore3.xml</bookstore>
</books>
```

The following query returns the bookstore elements:

```
/books/bookstore
```

Now suppose you pass this query to the document() function as follows:

```
document(/books/bookstore)
```

This query returns the root nodes of bookstore1.xml, bookstore2.xml, and bookstore3.xml.

XPath Quick Reference

This section includes the following topics:

- [“XPath Functions Quick Reference”](#) on page 708
- [“XPath Syntax Quick Reference”](#) on page 712
- [“XPath Abbreviations Quick Reference”](#) on page 713

See also [“Precedence of Query Operators”](#) on page 655.

XPath Functions Quick Reference

Table 91 lists the functions you can call in a query and provides short descriptions.

Table 91. XPath Function Quick Reference

<i>Function</i>	<i>Source</i>	<i>Returns</i>
<code>boolean()</code>	XPath	Boolean value that is the result of converting an object to a Boolean value. See “Converting an Object to Boolean” on page 678.
<code>ceiling()</code>	XPath	Number that is the smallest integer that is not less than a number you specify. See “Obtaining the Largest, Smallest, or Closest Number” on page 682.
<code>comment()</code>	XPath	Comment nodes. See “Obtaining Particular Types of Nodes By Using Node Tests” on page 694.
<code>concat()</code>	XPath	String that concatenates two or more strings you specify. See “Concatenating Strings” on page 673.
<code>contains()</code>	XPath	Nodes that contain the specified string. See “Searching for Strings” on page 670.
<code>count()</code>	XPath	Number of nodes in the <i>node-set</i> argument. See “Determining the Number of Nodes in a Collection” on page 700.
<code>current()</code>	XPath	Node for which the current template started its operation. See “Obtaining the Current Node for the Current XSLT Template” on page 702.
<code>document()</code>	XSLT	Root node of the specified document. See “Accessing Other Documents During Query Execution” on page 705.
<code>element-available()</code>	XSLT	Boolean value that indicates whether the specified element is supported by the XSLT processor. See “Determining If Functions Are Available” on page 702.
<code>false()</code>	XPath	false. See “Obtaining Boolean Values” on page 679.

Table 91. XPath Function Quick Reference

Function	Source	Returns
<code>floor()</code>	XPath	Number that is the largest integer that is not greater than a number you specify. See “Obtaining the Largest, Smallest, or Closest Number” on page 682.
<code>function-available()</code>	XSLT	Boolean value that indicates whether the specified function is supported by the XPath processor. See “Determining If Functions Are Available” on page 702.
<code>generate-id()</code>	XSLT	String that uniquely, temporarily, identifies a node. See “Generating Temporary IDs for Nodes” on page 705.
<code>id()</code>	XPath	Element whose <code>id</code> attribute value matches the specified value. See “Finding an Element with a Particular ID” on page 692.
<code>key()</code>	XSLT	Node whose key value matches the specified key. See “Finding an Element with a Particular Key” on page 703.
<code>lang()</code>	XPath	Boolean value that indicates whether the language of the node is the language you expect. See “Determining the Context Node Language” on page 679.
<code>last()</code>	XPath	Number of nodes in the context list. See “Determining the Number of Nodes in a Collection” on page 700.
<code>local-name()</code>	XPath	Local portion of the node name, excluding the prefix. See “Obtaining Namespace Information” on page 697.
<code>name()</code>	XPath	String that contains the tag name of the node, including namespace information, if any. See “Obtaining Namespace Information” on page 697.

Table 91. XPath Function Quick Reference

Function	Source	Returns
namespace-uri()	XPath	URI for the namespace of the node. See “Obtaining Namespace Information” on page 697.
node()	XPath	All nonattribute nodes. See “Obtaining Particular Types of Nodes By Using Node Tests” on page 694.
normalize-space()	XPath	String without leading or trailing white space. See “Normalizing Strings” on page 674.
not()	XPath	Boolean value that indicates the opposite of the specified Boolean value. See “Obtaining Boolean Values” on page 679.
number()	XPath	Number that is the result of converting the specified argument to a number. See “Converting an Object to a Number” on page 681.
position()	XPath	Position number of the node relative to the context node set. See “Finding a Particular Node” on page 688.
processing-instruction()	XPath	Processing instruction nodes. If you specify a literal argument, this function returns a processing instruction if its name matches the literal you specify. See “Obtaining Particular Types of Nodes By Using Node Tests” on page 694.
round()	XPath	Number that is the closest to the argument and is an integer. See “Obtaining the Largest, Smallest, or Closest Number” on page 682.
starts-with()	XPath	Boolean value that indicates if a string starts with a particular string. See “Finding Strings That Start with a Particular String” on page 676.
string()	XPath	String that is the result of converting some object to a string. See “Converting Objects to Strings” on page 675.

Table 91. XPath Function Quick Reference

Function	Source	Returns
string-length()	XPath	Number of characters in a string you specify. See “Determining the Number of Characters in a String” on page 673.
substring()	XPath	Substring that is in a particular position within its string. See “Finding Substrings by Position” on page 672.
substring-before()	XPath	Substring that appears before a string you specify. See “Finding Substrings That Appear Before Strings You Specify” on page 671.
substring-after()	XPath	Substring that appears after a string you specify. See “Finding Substrings That Appear After Strings You Specify” on page 672.
sum()	XPath	Number that is the sum of the values of the nodes in the specified set. See “Obtaining the Sum of the Values in a Node Set” on page 682.
system-property()	XSLT	Object that represents the specified property. See “Obtaining System Properties” on page 701.
translate()	XPath	String with some characters replaced by other characters. See “Replacing Characters in Strings with Characters You Specify” on page 674.
true()	XPath	true. See “Obtaining Boolean Values” on page 679.
unparsed-entity-uri()	XSLT	URI of an unparsed entity with the specified name. See “Obtaining the URI for an Unparsed Entity” on page 700.

XPath Syntax Quick Reference

This topic provides a quick reference for XPath expression syntax.

Axes

XPath provides the following axes:

- ancestor
- ancestor-or-self
- attribute
- child
- descendant
- descendant-or-self
- following
- following-sibling
- namespace
- parent
- preceding
- preceding-sibling
- self

Node Tests

XPath provides the following node tests:

- * selects all nodes of the specified name. For the attribute axis, attributes are selected. For the namespace axis, namespace nodes are selected. For all other axes, element nodes are selected.
- comment() selects all comment nodes.
- *element_name* selects all *element_name* nodes.
- node() selects all nodes.
- processing-instruction(["*some_literal*"]) selects all processing instructions. If *some_literal* is specified, processing-instruction() selects all processing instructions with *some_literal* as their name.
- text() selects all text nodes.

Filters

A filter specifies a constraint on a node set with respect to an axis to produce a new node set.

Location Steps

A location step has the following format:

```
AxisSpecifier::NodeTest[Filter][Filter]...
```

XPath Expression

An XPath expression has one of the following formats:

```
LocationStep[/LocationStep]...  
FunctionCall()[Filter]/LocationStep[/LocationStep]...  
(Expression)[Filter]/LocationStep[/LocationStep]...
```

A function call or an XPath expression in parentheses can appear only at the very beginning of an XPath expression. An expression in parentheses always returns a node set. Any function that appears at the beginning of an XPath location step expression must return a node set.

XPath Abbreviations Quick Reference

Table 92 defines the abbreviations you can use in XPath expressions:

Table 92. XPath Abbreviations Quick Reference

<i>Abbreviation</i>	<i>Description</i>
No axis is specified in a location step.	The <code>child</code> axis is assumed. For example, the following two XPath expressions both return the <code>para</code> children of <code>chapter</code> children of the context node: <code>chapter/para</code> <code>child::chapter/child::para</code>
@	The attribute axis. For example, the following two XPath expressions both return <code>para</code> children of the context node that have type attributes with a value of <code>warning</code> : <code>para[@type="warning"]</code> <code>child::para[attribute::type="warning"]</code>

Table 92. XPath Abbreviations Quick Reference

Abbreviation	Description
//	<p>The descendant-or-self axis. For example, the following two XPath expressions both return all para descendants of the context node:</p> <pre>//para /descendant-or-self::node()/child::para</pre> <p>However, it is important to note that the following two expressions are <i>not</i> equivalent:</p> <pre>/descendant::para[1] //para[1]</pre> <p>The first expression selects the first para element that is a descendant of the context node. The second expression selects each para descendant that is the first para child of its parent.</p>
.	<p>A single dot is the abbreviation for self::node(). This selects the context node. For example, the following two XPath expressions both return all para descendants of the context node:</p> <pre>./para self::node()/descendant-or-self::node()/child::para</pre>
..	<p>A double dot is the abbreviation for parent::node(). This selects the parent of the context node. For example, the following two XPath expressions both return the title children of the parent of the context node:</p> <pre>../title parent::node()/child::title</pre>

[Table 93](#) shows examples of abbreviations in XPath expressions

Table 93. Abbreviations in XPath Expressions

Example	Description
para	Selects the para children of the context node
*	Selects all element children of the context node
<i>node_test</i>	Evaluates all children of the context node and returns those that test true for the particular <i>node_test</i>
*/para	Selects all para grandchildren of the context node
para[1]	Selects the first para child of the context node

Table 93. Abbreviations in XPath Expressions

<i>Example</i>	<i>Description</i>
<code>para[<i>last()</i>]</code>	Selects the last <code>para</code> child of the context node
<code>/doc/chapter[5]/section[2]</code>	Selects the second section of the fifth chapter of the <code>doc</code> child of the context node
<code>para[@type="warning"]</code>	Selects <code>para</code> children of the context node that have <code>type</code> attributes with a value of <code>warning</code>
<code>para[@type="warning"][5]</code>	Selects the fifth <code>para</code> child of the context node that has a <code>type</code> attribute with a value of <code>warning</code>
<code>para[5][@type="warning"]</code>	Selects the fifth <code>para</code> child of the context node if that child has a <code>type</code> attribute with a value of <code>warning</code>
<code>chapter[title]</code>	Selects the chapter children of the context node that have one or more <code>title</code> children
<code>//para</code>	Selects all <code>para</code> descendants of the document root
<code>chapter//para</code>	Selects all <code>para</code> descendants of <code>chapter</code> children of the context node
<code>//olist/item</code>	Selects all <code>item</code> elements that have <code>olist</code> parents
<code>.</code>	Selects the context node
<code>./para</code>	Selects the <code>para</code> descendants of the context node
<code>..</code>	Selects the parent of the context node
<code>@*</code>	Selects all attributes of the context node
<code>@name</code>	Selects the <code>name</code> attribute of the context node
<code>../@name</code>	Selects the <code>name</code> attribute of the parent of the context node

Chapter 10 Working with XQuery in Stylus Studio



XQuery support is available only in Stylus Studio XML Enterprise Suite and Stylus Studio XML Professional Suite. Some features, like query plan, are available only in Stylus Studio XML Enterprise Suite.

Stylus Studio provides many features for working with XML Query (XQuery), including a graphical mapper that allows you to construct a query without writing any code, and tools to help you run and debug XQuery.



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the Introduction to the XQuery Mapper video](#).

You can learn more about other video demonstrations of the XQuery Mapper here: http://www.stylusstudio.com/learn_xquery.html#xquery_1.

This chapter covers the following topics:

- “Getting Started with XQuery in Stylus Studio” on page 718
- “An XQuery Primer” on page 723
- “Understanding FLWOR Expressions” on page 734
- “Building an XQuery Using the Mapper” on page 750
- “Working with the XQuery collection() Function” on page 777
- “Debugging XQuery Documents” on page 787
- “Creating an XQuery Scenario” on page 800
- “Using DataDirect XQuery® Execution Plans” on page 795
- “Generating XQuery Documentation” on page 810
- “Using XQuery to Invoke a Web Service” on page 816

- [“Generating Java Code for XQuery”](#) on page 819

Getting Started with XQuery in Stylus Studio

This section describes working with XQuery in Stylus Studio. It covers the following topics:

- [What is XQuery?](#) on page 718
- [What is an XQuery?](#) on page 719
- [The Stylus Studio XQuery Editor](#) on page 719

What is XQuery?

XML Query (XQuery) is the World Wide Web Consortium (W3C) language for querying XML. XQuery is a language developed by the W3C XML Query working group.

Example

The XQuery grammar allows you to define expressions like those shown in the following sample XQuery, R-Q2.xquery:

```
<result>
{
  for $i in document("items.xml")/items/item_tuple
  let $b := document("bids.xml")//bid_tuple[itemno = $i/itemno]
  where contains($i/description,"Bicycle")
  order by $i/itemno
  return
  <item_tuple>
    {
      {
        $i/itemno
        $i/description
        <high_bid>
          {
            max ( for $c in $b/bid return xs:decimal($c) )
          }
        </high_bid>
      }
    }
  </item_tuple>
}
</result>
```

This and other XQuery examples are provided in the Stylus Studio `examples\XQuery` directory.

Sources for Additional XQuery Information

See [An XQuery Primer](#) on page 723 if you are just getting started with XQuery. For more detailed information, including the formal W3C XQuery specification, visit <http://www.w3.org/XML/Query> (the W3C page for XML Query).

What is an XQuery?

In Stylus Studio, an XQuery is a document with a `.xquery` extension. Stylus Studio expects documents with this extension to contain a query expressed using the XML Query language.

The Stylus Studio XQuery Editor

In Stylus Studio, you use the XQuery editor's textual editor and graphical interfaces to work with XQuery. The XQuery editor, which consists of two tabs, **XQuery Source** and **Mapper**, is displayed any time the active document within Stylus Studio is an XQuery document. You can use either or both tabs to compose an XQuery.

By default, Stylus Studio gives new XQuery files a `.xquery` extension. You can save XQueries using any extension you choose. If you decide to use a different extension, use the **File Types** page of the **Options** dialog box to associate that extension with the XQuery editor.

XQuery Source Tab

You can use the **XQuery Source** tab to view, compose, preview, and debug your XQuery. For example, you can edit query text directly, set breakpoints, and debug your XQuery on this tab. The tab is divided into two panes:

- An editing pane, which shows the XQuery code, and
- A schema pane, which displays the schema of the source documents you are using to build your XQuery. You can hide the schema pane to view more of the XQuery code by clicking the show/hide button at the base of the splitter, which allows you to vary the relative width of the two panes.

Tip You can drag schema objects directly to the editing pane. This allows you to quickly create FLWOR and XPath expressions, for example, without writing any code or introducing typographical errors to the source.

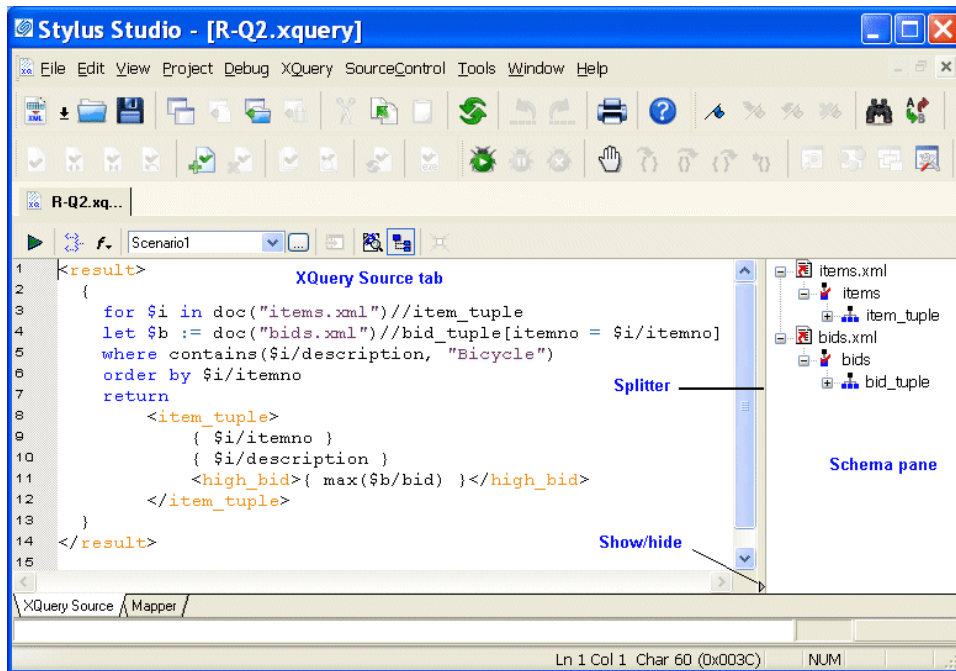



Figure 286. XQuery Source Tab

Stylus Studio's Sense:X automatic completion feature is supported for XQuery – Sense:X simplifies editing and helps ensure well-formed XML for queries you compose manually.

You can define other XQuery editor settings on the **Editor General** and **Editor Format** pages of the **Options** dialog box. (Click **Tools > Options**.)

You can preview the XQuery result by clicking the **Preview Result** button (). Results are displayed in the **Preview** window at the bottom of the XQuery editor, and, optionally, in any external application that you specify.

Mapper Tab

The **Mapper** tab provides an interface that allows you to compose and view your XQuery graphically.

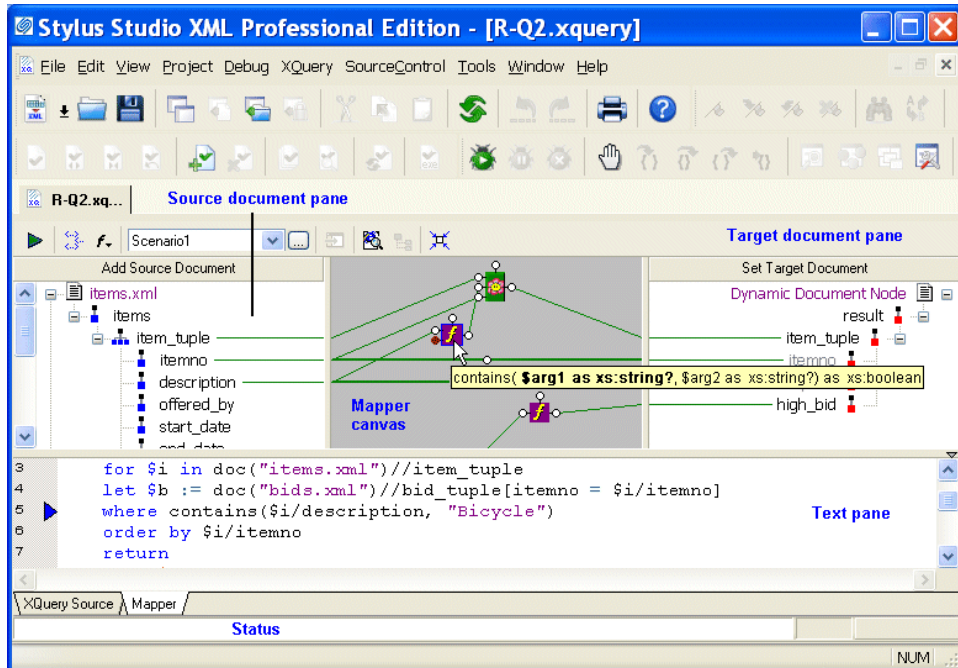



Figure 287. XQuery Editor Mapper Tab

The **Mapper** tab consists of these areas:

- Source document pane, in which you add one or more source documents.
- Target structure pane, in which you specify the structure of the result you want the XQuery to return.
- Mapper canvas, on which you can define conditions, functions, and operations for source document nodes to filter return values that are then mapped to the target node.
- Text pane. The text pane allows you to view the XQuery code while using the mapper. This is a great way to see how changes to the mapper affect the XQuery code, without the need to switch to the **XQuery Source** tab. Of course, the **XQuery Source** tab is available if you prefer working with the code using a full-page view. All views – **Mapper** tab, **XQuery Source** tab, and the text pane – are synchronized.

As with the **XQuery Source** tab, you can preview XQuery results from the **Mapper** tab by clicking the **Preview Result** button (). Debugging, however, can be performed from the **XQuery Source** tab only.

See “[Building an XQuery Using the Mapper](#)” on page 750 to learn more about the features of the XQuery editor **Mapper** tab.

XQuery Source and Mapper Tab Interaction

Changes made to an XQuery on the **Mapper** tab are reflected on the **XQuery Source** tab, and vice versa. For example, if you start writing your XQuery on the **XQuery Source** tab and then click the **Mapper** tab, Stylus Studio displays a graphic representation of your XQuery code. If you next edit the XQuery graphically (adding a function or a FLWOR block and mapping the return value to a node in the target structure, for example) and then return to the **XQuery Source** tab, you will see that Stylus Studio has updated the XQuery code based on your edits on the **Mapper** tab. Viewing the code on the **XQuery Source** tab that Stylus Studio creates based on actions performed on the **Mapper** tab can be a useful aid to learning XQuery syntax.

Note An incomplete XQuery artifact created on the **Mapper** tab is removed from the XQuery you are composing when you click the **XQuery Source** tab because it cannot be expressed in XQuery given its current definition. For example, imagine creating a FLWOR block that is not mapped to a node in the target structure. The FLWOR (pronounced “flower”) block appears on the **Mapper** tab, but Stylus Studio does not generate any code for it or display it on the **XQuery Source** tab, and when you return to the **Mapper** tab you will see that the FLWOR block has been removed.

An XQuery Primer

This XQuery primer was adapted from a whitepaper written by Dr. Michael Kay. You can read the original document on the [Stylus Studio Web site](#). This primer covers the following topics:

- “What is XQuery For?” on page 723
- “Your First XQueries” on page 723
- “Accessing XML Documents with XQuery” on page 724
- “XQuery and XPath” on page 726
- “Introduction to FLWOR Expressions” on page 730
- “Generating XML Output with XQuery” on page 731
- “Accessing Databases with XQuery” on page 733

What is XQuery For?

XQuery was devised primarily as a query language for data stored in XML form. So its main role is to get information out of XML databases — this includes relational databases that store XML data, or that present an XML view of the data they hold.

Some people are also using XQuery for manipulating free-standing XML documents, for example, for transforming messages passing between applications. In that role XQuery competes directly with XSLT, and which language you choose is largely a matter of personal preference.

In fact, some people like XQuery so much that they are even using it for rendering XML into HTML for presentation. That's not really the job XQuery was designed for, and there are other technologies, like XSLT, that are better suited for this purpose, but once you get to know a tool, you tend to find new ways of using it.

Your First XQueries

Try a few simple examples to get acquainted with XQuery. Start Stylus Studio, and open a new XQuery document (**File > New > XQuery File**). Save the file now (if you do not, Stylus Studio will prompt you to save it when you preview your first XQuery).

Type the following in the XQuery Editor:

```
"Hello, world!"
```

Now, click the **Preview Result** button (▶), and Stylus Studio displays the result of this XQuery in the **Preview** window:

```
"Hello, world!"
```

Enter a simple equation ($2+2$) and click the **Preview Result** button (▶). Predictably, the result is:

```
4
```

Finally, try the XQuery function `current-time()` and click the **Preview Result** button (▶):

```
11:27:38Z
```

Your results, will vary based on several conditions. For example, the [XQuery processor](#) you use to execute the XQuery will affect the precision of the time value (fractions of seconds), and the time zone (here, shown as *Z*) is determined by how your system is configured.

None of these is a very useful query on its own, but within a query language you need to be able to perform little calculations and XQuery has this covered. Further, XQuery is designed so that expressions can be fully nested – that is, any expression can be used within any other expression, provided that it delivers a value of the right type – and this means that expressions that are primarily intended for selecting data within a *where* clause can also be used as free-standing queries in their own right.

Accessing XML Documents with XQuery

Though XQuery is capable of handling mundane tasks like those described in the previous section, it is designed to access XML data. Right now, we will look at some simple queries that require an XML document as their input. For this purpose, we will use `videos.xml`, which is installed with Stylus Studio in the `\examples\VideoCenter` directory. You can also find a copy of this XML document on the [Stylus Studio Web site](#).

XQuery allows you to access the file directly from either of these locations, using a suitable URL as an argument for its `doc()` function. If you wanted to retrieve and display

the entire file from your Stylus Studio installation directory, your `doc()` might look like this:

```
doc('file:///c:/Program%20Files/Stylus%20Studio%202007%20XML%20Professiona  
l%20Suite/examples/VideoCenter/videos.xml')
```

To fetch this document from the Stylus Studio Web site, you would need a `doc()` like this:

```
doc('http://www.stylusstudio.com/examples/videos.xml')
```

(The latter `doc()` function will work only if you are online; and if you are behind a corporate firewall you might have to modify your Java configuration to make it work.)

Handling URLs

URLs like those used in the previous example can be a bit unwieldy, but there are some shortcuts you can use.

- In Stylus Studio, you can specify the source document as the **Main Input** on the **General** tab of the [Scenario Properties](#) dialog box. Once you browse to the appropriate file and select it, you can refer to it in your XQuery code as simply “.” (dot).
- If you are working directly with a command line processor such as Saxon, you can copy the file locally (`c:\xquery\videos.xml`, for example) and work with it from that location. Once you have done this, you can use the command line option `-s c:\xquery\videos.xml` and again be able to refer to the input document in your XQuery code as “.” (dot).

The videos.xml Document

The `videos.xml` document contains a number of sections: `video_template`, `actors`, and `videos`. You might want to open this document in the XML Editor to get acquainted with it if you are not already familiar with it.

XQuery and XPath

We can access the actors section of `videos.xml` using this expression: `./actors`. When we execute this XQuery we get the following result:

```
<actors>
  <actor id="00000015">Anderson, Jeff</actor>
  <actor id="00000030">Bishop, Kevin</actor>
  <actor id="0000000f">Bonet, Lisa</actor>
  <actor id="00000024">Bowz, Eddie</actor>
  <actor id="0000002d">Curry, Tim</actor>
  <actor id="00000033">Dullea, Keir</actor>
  <actor id="00000042">Fisher, Carrie</actor>
  <actor id="00000006">Ford, Harrison</actor>
  <actor id="00000045">Foster, Jodie</actor>
  ...etc...
</actors>
```

That was our first “real” query. If you are familiar with XPath, you might recognize that all the queries so far have been valid XPath expressions. We have used a couple of functions – `current-time()` and `doc()` – that might be unfamiliar because they are new in XPath 2.0, which is still only a draft; but the syntax of all the queries so far is plain XPath syntax. In fact, the XQuery language is designed so that every valid XPath expression is also a valid XQuery query.

This means we can write more complex XPath expressions like this one:

```
./actors/actor[ends-with(., 'Lisa')]
```

which gives us this output:

```
<actor id="0000000f">Bonet, Lisa</actor>
<actor id="0000001b">Spoonhauer, Lisa</actor>
```

Different systems might display this output in different ways. Technically, the result of this query is a sequence of two element nodes in a tree representation of the source XML document, and there are many ways a system might choose to display such a sequence on the screen. Stylus Studio gives you the choice of a text view, like the one shown above,

and a tree view: you use the buttons next to the Preview window to switch from one to the other. Here is what the tree view looks like:

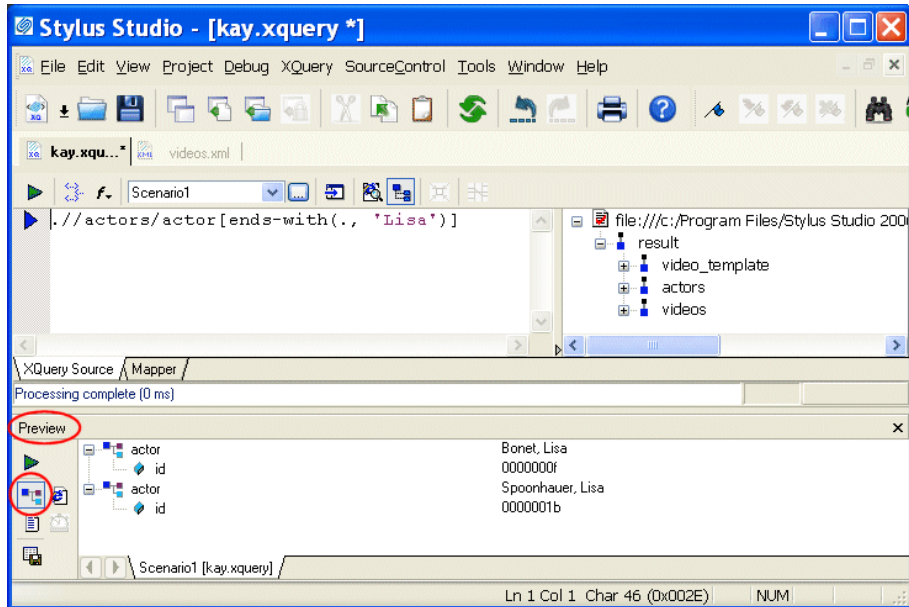


Figure 288. XQuery Preview Window Tree View

This example used another function – `ends-with()` – that's new in XPath 2.0. We are calling it inside a predicate (the expression between the square brackets), which defines a condition that nodes must satisfy in order to be selected. This XPath expression has two parts: a path `./actors/actor` that indicates which elements we are interested in, and a predicate `[ends-with(., 'Lisa')]` that indicates a test that the nodes must satisfy. The predicate is evaluated once for each selected element; within the predicate, the expression `."` (dot) refers to the node that the predicate is testing, that is, the selected actor.

The `/` in the path informally means “go down one level”, while the `./` means “go down any number of levels”. If the path starts with `./` or `./` you can leave out the initial `."` (This assumes that the selection starts from the top of the tree, which is always the case in our examples.) You can also use constructs like `../` to go up one level, and `/@id` to select an attribute. Again, this is all familiar if you already know XPath.

XPath is capable of doing some pretty powerful selections, and before we move on to XQuery proper, let us look at a more complex example. Suppose we want to find the titles

of all the videos featuring an actor whose first name is Lisa. If we look at the `videos.xml` file, we see that each video in the file is represented by a video element like this one:

```
<video id="647599251">
  <studio></studio>
  <director>Francesco Rosi</director>
  <actorRef>916503211</actorRef>
  <actorRef>916503212</actorRef>
  <title>Carmen</title>
  <dvd>18</dvd>
  <laserdisk></laserdisk>
  <laserdisk_stock></laserdisk_stock>
  <genre>musical</genre>
  <rating>PG</rating>
  <runtime>125</runtime>
  <user_rating>4</user_rating>
  <summary>A fine screen adaptation of Bizet's popular opera.</summary>
  <details>Placido Domingo does it again, this time in Bizet's popular opera.</details>
  <vhs>15</vhs>
  <beta_stock></beta_stock>
  <year>1984</year>
  <vhs_stock>88</vhs_stock>
  <dvd_stock>22</dvd_stock>
  <beta></beta>
</video>
```

The query required to provide the results we desire is written like this:

```
//video[actorRef=//actors/actor[ends-with(., 'Lisa')]]/@id]/title
```

Again, this is pure XPath (and therefore a valid XQuery). You can read it from left-to-right as:

- Start at the implicitly-selected document (`videos.xml`)
- Select all the `<video>` elements at any level
- Choose those that have an `actorRef` element whose value is equal to one of the values of the following:
 - Select all the `<actors>` elements at any level
 - Select all their `<actor>` child elements
 - Select the element only if its value ends with 'Lisa'
 - Select the value of the `id` attribute
- Select the `<title>` child element of these selected `<video>` elements

When run against our source document, the result of this XQuery is:

```
<title>Enemy of the State</title>
<title>Clerks</title>
```

Many people find that at this level of complexity, XPath syntax gets rather mind-boggling. In fact, this example just about stretches XPath to its limits. For this kind of query, and for anything more complicated, XQuery syntax comes into its own. But it is worth remembering that there are many simple things you can do with XPath alone, and that every valid XPath expression is also valid in XQuery.

XPath Query Editor

Stylus Studio provides a built-in XPath Query Editor in its XML Editor that allows you to visually edit and test complex XPath expressions, and it supports both version 1.0 and 2.0.

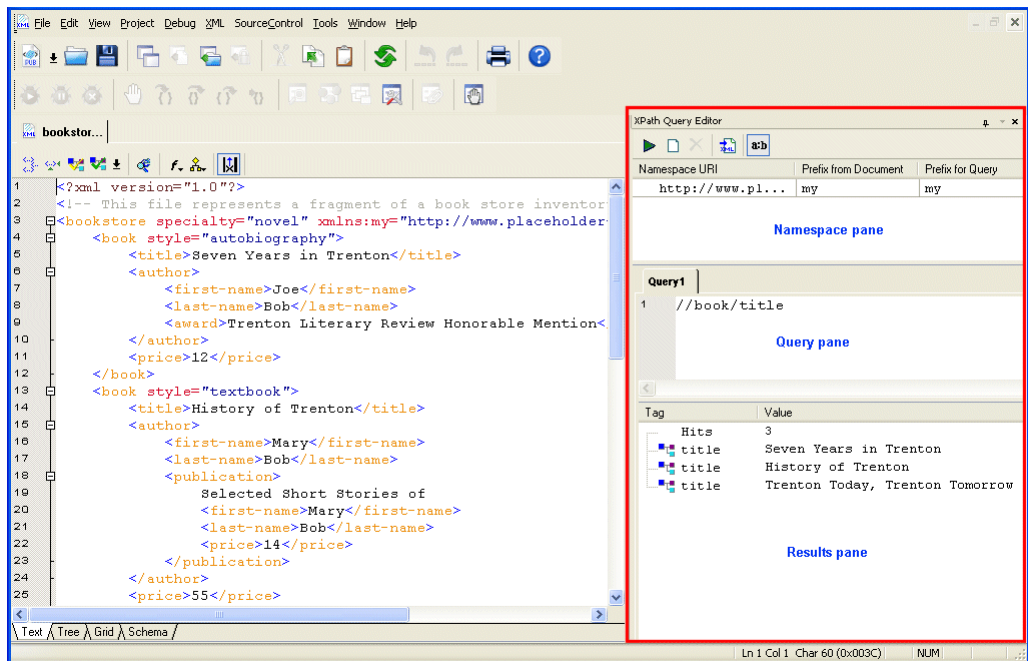


Figure 289. XPath Query Editor

See [“Using the XPath Query Editor”](#) on page 632 to learn more about using this tool.

Introduction to FLWOR Expressions

If you have used SQL, then you will have recognized the last example as a join between two tables – the videos table and the actors table. Join queries are not quite the same in XML, because the data is hierarchic rather than tabular, but XQuery allows you to write join queries in a similar way to the familiar SQL approach. The equivalent of the SQL SELECT expression is called the FLWOR expression, named after its five clauses: for, let, where, order by, return. Here is the last example (all videos with an actor named ‘Lisa’, rewritten this time as a FLWOR expression).

```
let $doc := .
for $v in $doc//video,
  $a in $doc//actors/actor
where ends-with($a, 'Lisa')
  and $v/actorRef = $a/@id
return $v/title
```

When we run this XQuery, we get the same result as before.

Let’s examine the FLWOR expression:

- The `let` clause simply declares a variable. We included this here because when we deploy the query we might want to set this variable differently; for example, we might want to initialize it to `doc('videos.xml')`, or to the result of some complex query that locates the document in a database.
- The `for` clause defines two range variables: one processes all the videos in turn, the other processes all the actors in turn. Taken together, the FLWOR expression is processing all possible pairs of videos and actors.
- The `where` clause then selects those pairs that we are actually interested in. We are only interested if the actor appears in that video, and we are only interested if the actor’s name ends in ‘Lisa’.
- Finally the return clause tells the system what information we want to get back. In this case we want the title of the video.

If you have been following very closely, you might have noticed one little XPath trick that we retained in this query: most videos will feature more than one actor (though this particular database does not attempt to catalog the bit-part players). The expression `$v/actorRef` therefore selects several elements. The rules for the `=` operator in XPath (and therefore also in XQuery) are that it compares everything on the left with everything on the right and returns true if there is at least one match. In effect, it is doing an implicit join.

If you want to avoid exploiting this feature, and you want to write your query in a more classically relational form, you could express it as follows:

```
let $doc := .
for $v in $doc//video,
    $va in $v/actorRef,
    $a in $doc//actors/actor
where ends-with($a, 'Lisa')
    and $va eq $a/@id
return $v/title
```

This time we used a different equality operator, `eq`, which follows more conventional rules than `=` does: it strictly compares one value on the left with one value on the right. (But like comparisons in SQL, it has special rules to handle the case where one of the values is absent.)

What about the `O` in FLWOR? That is there so you can get the results in sorted order. Suppose you want the videos in order of their release date. Here's the revised query:

```
let $doc := .
for $v in $doc//video,
    $a in $doc//actors/actor
where ends-with($a, 'Lisa')
    and $v/actorRef = $a/@id
order by $v/year
return $v/title
```

If you are wondering why FLWOR is not really a LFWOR expression: the `for` and `let` clauses can appear in any order, and you can have any number of each. To learn more about the FLWOR expression, see [Understanding FLWOR Expressions](#) on page 734.

Generating XML Output with XQuery

So far all the queries we have written have selected nodes in the source document. We have shown the results as if the system copies the nodes to create some kind of result document, and if you execute the XQuery in Stylus Studio or run Saxon from the command line that is exactly what happens. But this is simply a default mode of execution. In a real application you want control over the form of the output document, which might well be the input to another application – perhaps the input to an XSLT transformation or even another query.

XQuery allows the structure of the result document to be defined using an XML-like notation. Here is an example that fleshes out our previous query with some XML markup:

```
declare variable $firstName as xs:string external;
<videos featuring="{ $firstName}">
{
  let $doc := .
  for $v in $doc//video,
    $a in $doc//actors/actor
  where ends-with($a, $firstName)
  and $v/actorRef = $a/@id
  order by $v/year
  return
    <video year="{ $v/year}">
      { $v/title }
    </video>
}
</videos>
```

We have also changed the query so that the actor's first name is now an externally defined parameter. This makes the query reusable. The way parameters are supplied varies from one XQuery processor to another. In Stylus Studio, select **XQuery > Scenario Properties**; click the **Parameter Values** tab, and Stylus Studio provides an area to specify values for any variables defined in the XQuery.

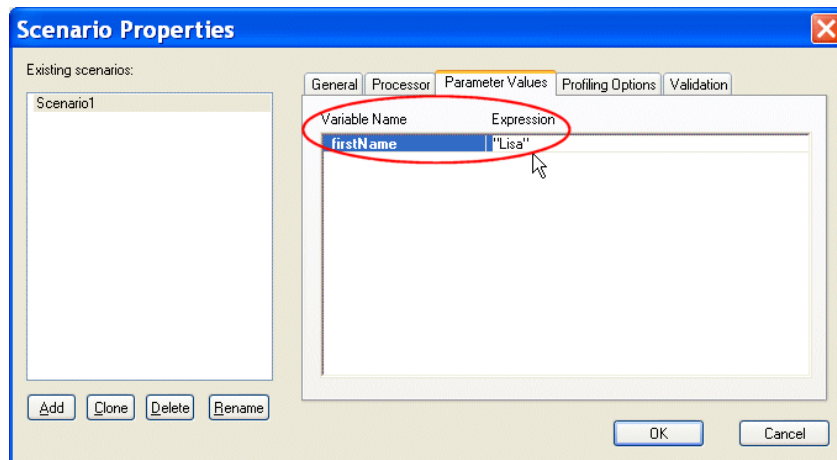


Figure 290. Entering Values for XQuery Variables

Enter “Lisa”, in quotes (Stylus Studio expects an expression, so if the quotes are omitted, this value would be taken as a reference to an element named <Lisa>).

If instead you are running Saxon from the command line, you can enter:

```
java net.sf.saxon.Query sample.xquery firstName=Lisa
```

Either way, our XQuery returns the following result:

```
<videos featuring="Lisa">
  <video year="1999">
    <title>Enemy of the State</title>
  </video>
  <video year="1999">
    <title>Clerks</title>
  </video>
</videos>
```

As you might recall from our previous XQuery, this version of the XQuery is not especially well-designed as it returns videos featuring different actresses named Lisa. Take some time and modify the XQuery to see if you can improve it.

Accessing Databases with XQuery

At the start of this section, we stated that the main purpose of XQuery is to extract data from XML databases, but all our examples have used a single XML document as input.

People sometimes squeeze a large data set (for example, a corporate phone directory) into a single XML document, and process it as a file without the benefit of any database system. While there are preferable alternatives, if the data volumes do not go above a few megabytes and the transaction rate is modest, then XML-document-as-database is a perfectly feasible storage mechanism. In other words, the examples in this section are not totally unrealistic.

If you have got a real database, however, the form of the queries used in this section will not need to change all that much from these examples. Instead of using the `doc()` function (or simply “.”) to select a document, you are likely to call the `collection()` function to open a database, or a specific collection of documents within a database. The actual way collections are named is likely to vary from one database system to another. The result of the XQuery `collection()` function is a set of documents (more strictly, a sequence of documents, but the order is unlikely to matter), and you can process this using XPath expressions or FLWOR expressions in just the same way as you address a single document.

There is a lot more to databases than doing queries, of course. Each product has its own ways of setting up the database, defining schemas, loading documents, and performing maintenance operations such as backup and recovery. XQuery currently handles only one

small part of the job. In the future it is also likely to have an update capability, but in the meantime each vendor is defining his own.

One particularly nice feature of XQuery is that it has the potential to combine data from multiple databases (and freestanding XML documents). DataDirect XQuery[®], which supports access to Oracle, DB2, SQL Server, and Sybase is one product that addresses this need.

Understanding FLWOR Expressions

This XQuery primer was adapted from a whitepaper written by Dr. Michael Kay. You can read the original document on the [Stylus Studio Web site](#). This section covers the following topics:

- “Simple XQuery FLWOR Expressions” on page 734
- “The Principal Parts of an XQuery FLWOR Expression” on page 735
- “Other Parts of the XQuery FLWOR Expression” on page 747
- “Grouping” on page 749

Simple XQuery FLWOR Expressions

The simplest XQuery FLWOR expression might be something like this:

```
for $v in $doc//video return $v
```

This returns all of the video elements in \$doc.

We can add a bit of substance by adding XQuery `where` and `return` clauses:

```
for $v in $doc//video
where $v/year = 1999
return $v/title
```

This returns all of the titles of videos released in 1999.

If you know SQL, that XQuery probably looks reassuringly similar to the equivalent SQL statement:

```
SELECT v.title
FROM video v
WHERE v.year = 1999
```

And if you know XPath, you might be wondering why our XQuery cannot be written as this:

```
$doc//video[year=1999]/title
```

Well, you can. This [XPath expression](#) is completely equivalent to the FLWOR expression above, and furthermore, it is a legal XQuery query. In fact, every legal XPath expression is also legal in XQuery. Thus the first query in this section can be written as:

```
$doc//video
```

Which style you prefer seems to depend on where you are coming from: if you have been using XML for years, especially XML with a deep hierarchy as found in “narrative” documents, then you will probably be comfortable with path expressions. But if you are more used to thinking of your data as representing a table, then the FLWOR style might suit you better.

As you will see, FLWOR expressions are a lot more powerful than path expressions when it comes to doing joins. But for simple queries, the capabilities overlap and you have a choice. Although it might be true that in SQL every query is a SELECT statement, it is not so that in XQuery every query has to be a FLWOR expression.

The Principal Parts of an XQuery FLWOR Expression

The name FLWOR comes from the five clauses that make up a FLWOR expression: `for`, `let`, `where`, `order by`, and `return`. Most of these clauses are optional: the only clause that is always present is the XQuery `return` clause (though there must be at least one XQuery `for` or `let` clause as well). To see how FLWOR expressions work, we will build up our understanding one clause at a time.

F is for For

The behavior of the `for` clause is fairly intuitive: it iterates over an input sequence and calculates some value for each item in that sequence, returning a sequence obtained by concatenating the results of these calculations. In simple cases there is one output item for every input item. So, this FLWOR expression:

```
for $i in (1 to 10)
return $i * $i
```

returns the sequence (1, 4, 9, 16, 25, 36, 49, 64, 81, 100). In this example, the input items are simple numbers, and the output items are also simple numbers. Numbers are an example of what XQuery calls atomic values; other examples are strings, dates, booleans, and URIs. But the XQuery data model allows sequences to contain XML nodes as well as atomic values, and the `for` expression can work on either.

Here is an example that takes nodes as input, and produces numbers as output. It counts the number of actors listed for each video in a data file:

```
for $v in //video
return count($v/actorRef)
```

You can run this in Stylus Studio or in Saxon, using the example `videos.xml` file as input. (Tips on setting up these tools are described in the previous section, “[An XQuery Primer](#)” on page 723.) Here's the output from Stylus Studio:

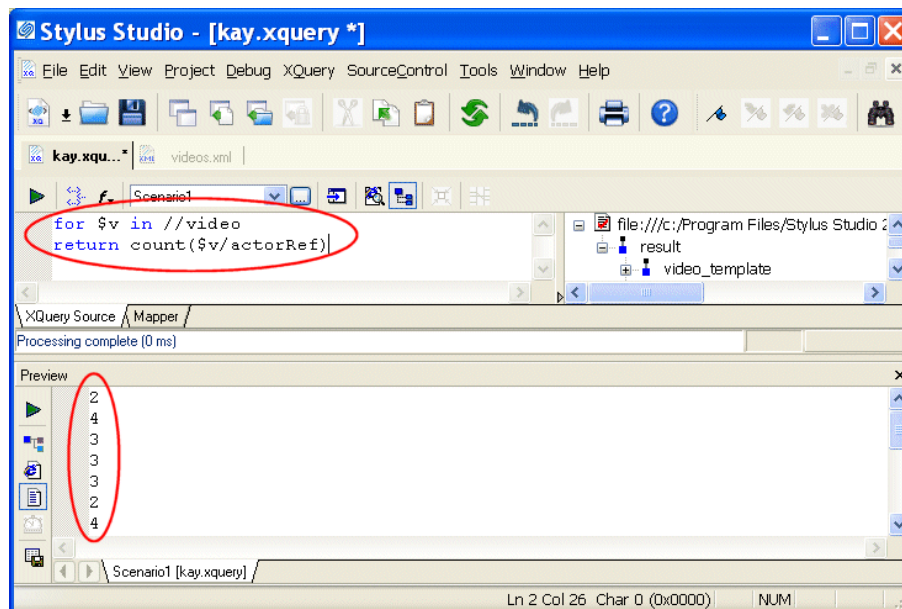


Figure 291. FOR Clause in a FLWOR Expression

The **Preview** window shows the result: a rather mundane sequence of numbers (2, 4, 3, 3, 3...).

This gives us a good opportunity to point out that a FLWOR expression is just an expression, and you can use it anywhere an expression is allowed: it doesn't have to be at

the top level of the query. There is a function, `avg()`, to compute the average of a sequence of numbers, and we can use it here to find the average number of actors listed for each of the movies in our data file:

```
avg(
  for $v in //video
  return count($v/actorRef)
)
```

The answer is 2.2941176 and a bit – the number of decimal places shown will depend on the XQuery processor that you use. If you are only interested in the answer to two decimal places, try:

```
round-half-to-even(
  avg(
    for $v in //video
    return count($v/actorRef)
  ),
  2)
```

which gives a more manageable answer of 2.29. (The strange name `round-half-to-even` is there to describe how this function does rounding: a value such as 2.145 is rounded to the nearest even number, in this case 2.14.) This is all designed to demonstrate that XQuery is a functional language, in which you can calculate a value by passing the result of one expression or function into another expression or function. Any expression can be nested inside any other, and the FLWOR expression is no exception.

If you are coming from SQL, your instinct was probably to try and do the averaging and rounding in the `return` clause. But the XQuery way is actually much more logical. The `return` clause calculates one value for each item in the input sequence, whereas the `avg()` function applies to the result of the FLWOR expression as a whole.

As with some of the examples in [“An XQuery Primer”](#) on page 723, XPath 2.0 allows you to write this example using path expressions alone if you prefer:

```
round-half-to-even(avg(//video/count(actorRef)), 2)
```

We have seen a `for` expression that produces a sequence of numbers from another sequence of numbers, and we have seen one that produces a sequence of numbers from a sequence of selected nodes. We can also turn numbers into nodes: the following query selects the first five videos.

```
for $i in 1 to 5 return (//video)[ $i ]
```

And we can get from one sequence of nodes to another sequence of nodes. This example shows all the actors that appear in any video:

```
for $actorId in //video/actorRef
return //actors/actor [@id=$actorId]
```

In fact, this last example probably represents the most common kind of `for` expression encountered, but we introduced it last to avoid leaving you with the impression that it is the only kind there is.

Once again, you could write this example as an XPath expression:

```
//actors/actor [@id="//video/actorRef]
```

However, this time the two expressions are not precisely equivalent. Try them both in Stylus Studio: the FLWOR expression produces a list containing 38 actors, while the list produced by the path expression contains only 36. The reason is that path expressions eliminate duplicates, and FLWOR expressions do not. Two actors are listed twice because they appear in more than one video.

The FLWOR expression and the “/” operator in fact perform quite similar roles: they apply a calculation to every item in a sequence and return the sequence containing the results of these calculations. There are three main differences between the constructs:

- The `for` expression defines a variable `$v` that is used in the `return` clause to refer to each successive item in the input sequence; a path expression instead uses the notion of a context item, which you can refer to as “.” In this example, `//video` is short for `./root()//video`, so the reference to the context item is implicit.
- With the “/” operator, the expression on the left must always select *nodes* rather than atomic values. In the earlier example `//video/count(actorRef)`, the expression on the right returned a number – that’s a new feature in XPath 2.0 – but the left-hand expression must still return nodes.
- When a path expression selects nodes, they are always returned in document order, with duplicates removed. For example, the expression `$doc//section//para` will return each qualifying `<para>` element exactly once, even if it appears in several nested `<section>` elements. If you used the nearest-equivalent FLWOR expression, `for $s in $doc//section return $s//para`, then a `<para>` that appears in several nested sections would appear several times in the output, and the order of `<para>` elements in the output will not necessarily be the same as their order in the original document.

The `for` clause really comes into its own when you have more than one of them in a FLWOR expression. We will explore that a little later, when we start looking at joins. But first, let's take a look at the other clauses: starting with `let`.

L is for Let

The XQuery `let` clause simply declares a variable and gives it a value:

```
let $maxCredit := 3000
let $overdrawnCustomers := //customer[overdraft > $maxCredit]
return count($overdrawnCustomers)
```

Hopefully the meaning of that is fairly intuitive. In fact, in this example you can simply replace each variable reference by the expression that provides the expression's value. This means that the result is the same as this:

```
count(//customer[overdraft > 3000])
```

In a `for` clause, the variable is bound to each item in the sequence in turn. In a `let` clause, the variable only takes one value. This can be a single item or a sequence (there is no real distinction in XQuery – an item is just a sequence of length one). And of course the sequence can contain nodes, or atomic values, or (if you really want) a mixture of the two.

In most cases, variables are used purely for convenience, to simplify the expressions and make the code more readable. If you need to use the same expression more than once, then declaring a variable is also a good hint to the XQuery processor to only do the evaluation once.

In a FLWOR expression, you can have any number of `for` clauses, and any number of `let` clauses, and they can be in any order. For example (returning to the `videos.xml` data again), you can do this:

```
for $genre in //genre/choice
let $genreVideos := //video[genre = $genre]
let $genreActorRefs := $genreVideos/actorRef
for $actor in //actor[@id = $genreActorRefs]
return concat($genre, ":", $actor)
```

To understand this, just translate it into English:

For each choice of genre, let's call the set of videos in that genre `$genreVideos`. Now let's call the set of references to all the actors in all those videos `$genreActorRefs`.

For each actor whose ID is equal to one of the references in `$genreActorRefs`, output

a string formed by concatenating the name of the genre and the name of the actor, separated by a colon.

Here is the result in Stylus Studio:

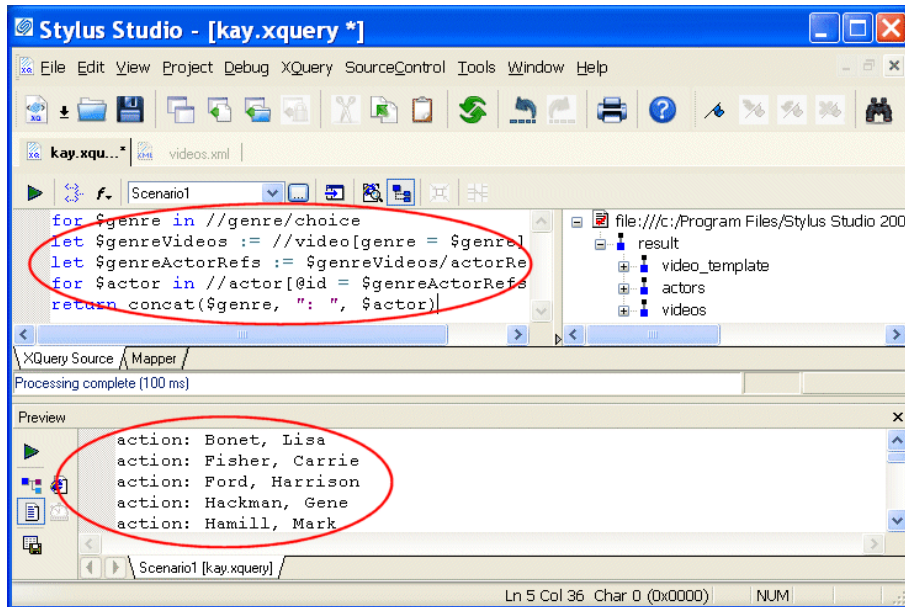


Figure 292. LET Clause in a FLWOR Expression

As a quick aside, the Stylus Studio XQuery Mapper allows you to visually map from one or more XML input documents to any target output format. In a nutshell – click on the **Mapper** tab on the bottom of the XQuery Editor. Next, click **Add Source Document** and

add your source document(s). Our last XQuery would look like this in the XQuery Mapper:

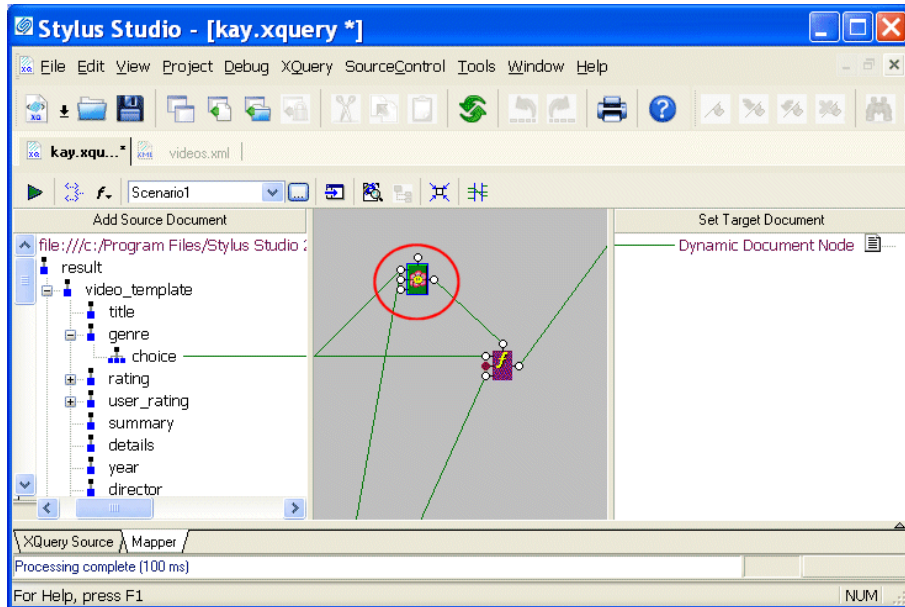


Figure 293. Simple FLWOR Shown in XQuery Mapper

The FLWOR block is graphically represented as a function block with three input ports going into it on the left (For, Where, Order By), a flow control port on the top, and an output port on the right. As you draw your XML mappings, Stylus Studio writes the XQuery code; similarly, you can edit the XQuery code manually and Stylus Studio which will update the graphical model – both views of the XQuery are kept synchronized. See [“Building an XQuery Using the Mapper”](#) on page 750 for more information on the Mapper module.

One important thing to note about variables in XQuery (you can skip this if you already know XSLT, because the same rule applies there): variables cannot be updated. This means you cannot write something like `let $x := $x+1`. This rule might seem very strange if you are expecting XQuery to behave in the same way as procedural languages such as JavaScript. But XQuery is not that kind of language, it is a declarative language and works at a higher level. There are no rules about the order in which different expressions are executed (which means that the little yellow triangle that shows the current execution point in the Stylus Studio XQuery debugger and XSLT debugger can sometimes behave in surprising ways), and this means that constructs whose result would

depend on order of execution (like variable assignment) are banned. This constraint is essential to give optimizers the chance to find execution strategies that can search vast databases in fractions of a second. Most XSLT users (like SQL users before them) have found that this declarative style of programming grows on you. You start to realize that it enables you to code at a higher level: you tell the system what results you want, rather than telling it how to go about constructing those results.

You might ask yourself at this point, Isn't a variable being updated when we write something like the following?

```
for $v in //video
let $x := xs:int($v/runtime) * xdt:dayTimeDuration("PT1M")
return concat($v/title, ": ",
             hours-from-duration($x), " hour(s) ",
             minutes-from-duration($x), " minutes")
```

(This query shows the running time of each video. It first converts the stored value from a string to an integer, then multiplies it by one minute (PT1M) to get the running time as a duration, so that it can extract the hours and minutes components of the duration. Try it.)

Here the variable `$x` has a different value each time around the XQuery for loop. This feels a bit like an update. Technically though, each time round the for loop you are creating a new variable with a new value, rather than assigning a new value to the old variable. What you cannot do is to accumulate a value as you go around the loop. Try doing this to see what happens:

```
let $totalDuration := 0
for $v in //video
let $totalDuration := $totalDuration + $v/runtime
return $totalDuration
```

The result is not a single number, but a sequence of numbers, one for each video. This example is actually declaring two separate variables that happen to have the same name. You are allowed to use the same variable name more than once, but this is probably not a good idea, because it will only get your readers confused. You can see more clearly what this query does if we rename one of the variables.

```
let $zero := 0
for $v in //video
let $totalDuration := $zero + $v/runtime
return $totalDuration
```

which is the same as this:

```
for $v in //video
return 0 + $v/runtime
```

Hopefully it is now clear why this returns a sequence of numbers rather than a single total. The correct way to get the total duration is to use the `sum` function:

```
sum(//video/runtime).
```

W is for Where

The XQuery `where` clause in a FLWOR expression performs a very similar function to the `WHERE` clause in a SQL `select` statement: it specifies a condition to filter the items we are interested in. The `where` clause in a FLWOR expression is optional, but if it appears it must only appear once, after all the `for` and `let` clauses. Here is an example that restates one of our earlier queries, but this time using a `where` clause:

```
for $genre in //genre/choice
for $video in //video
for $actorRefs in $video/actorRef
for $actor in //actor
where $video/genre = $genre
    and $actor/@id = $actorRefs
return concat($genre, " ", $actor)
```

This style of coding is something that SQL users tend to be very comfortable with: first define all the tables you are interested in, then define a `WHERE` expression to define all the restriction conditions that select subsets of the rows in each table, and join conditions that show how the various tables are related.

Although many users seem to find that this style comes naturally, an alternative is to do the restriction in a predicate attached to one of the `for` clauses, like this:

```
for $genre in //genre/choice
for $video in //video[genre = $genre]
for $actorRefs in $video/actorRef
for $actor in //actor[@id = $actorRefs]
return concat($genre, " ", $actor)
```

Perhaps there is a balance between the two; you will have to find the style that suits you best. With some XQuery processors one style or the other might perform better (and with Stylus Studio, you can easily create multiple XQuery scenarios that execute the same code but use different XQuery processors), but a decent optimizer is going to treat the two forms as equivalent.

Do remember that in a predicate, you select the item that you are testing relative to the context node, while in the `where` clause, you select it using a variable name. A bare name such as `genre` is actually selecting `./child:genre` – that is, it is selecting a child of the context node, which in this case is a `<video>` element. It is very common to use such expressions in predicates, and it is very uncommon to use them (except by mistake!) in the `where` clause. If you use a schema-aware processor like Saxon, then you might get an error message when you make this mistake; in other cases, it is likely that the condition will not select anything. The `where` condition will therefore evaluate to false, and you will have to puzzle out why your result set is empty.

O is for Order By

If there is no `order by` clause in a FLWOR expression, then the order of the results is as if the `for` clauses defined a set of nested loops. This does not mean they actually have to be evaluated as nested loops, but the result has to be the same as if they were. That is an important difference from SQL, where the result order in the absence of any explicit sorting is undefined. In fact, XQuery defines an alternative mode of execution, unordered mode, which is similar to the SQL rules. You can select this in the query prolog, and the processor might even make it the default (this is most likely to happen with products that use XQuery to search a relational database). Some products (Stylus Studio and Saxon among them) give you exactly the same result whether or not you specify unordered mode – since the XQuery specification says that in unordered mode anything goes, that is perfectly acceptable.

Often however you want the query results in sorted order, and this can be achieved using the `order by` clause. Let's sort our videos in ascending order of year, and within that in decreasing order of the user rating:

```
for $x in //video
order by $x/year ascending, number($x/user-rating) descending
return $x/title
```

Note that we have not actually included the sort keys in the data that we are returning (which makes it a little difficult to verify that it is working properly; but it is something you might well want to do in practice). We have explicitly converted the user-rating to a number here to use numeric sorting: this makes sure that 10 is considered a higher rating than 2. This is not necessary if the query is schema-aware, because the XQuery processor then knows that user-rating is a numeric field.

Ordering gets a little complicated when there is more than one `for` clause in the FLWOR expression. Consider this example:

```
for $v in //video
for $a in //actor
where $v/actorRef = $a/@id
order by $a, $v/year
return concat($a, ":", $v/title)
```

To understand this we have to stop thinking about the two `for` clauses as representing a nested loop. We cannot compute all the result values and then sort them, because the result does not contain all the data used for sorting (it orders the videos for each actor by year, but only shows their titles). In this case we could imagine implementing the order specification by rearranging the `for` clauses and doing a nested loop evaluation with a different order of nesting; but that doesn't work in the general case. For example, it wouldn't work if the `order by` clause changed to:

```
order by substring-after($a, ","),
        $v/year,
        substring-before($a, ",")
```

to sort first on the surname, then on the year, then on the first name (admittedly, nonsensical coding, but we show it only to illustrate that it is allowed).

The XQuery specification introduces a concept of tuples, borrowed from the relational model, and describes how the sort works in terms of creating a sequence of tuples containing one value for each of the variables, and then sorting these notional tuples.

R is for Return

Every XQuery FLWOR expression has a `return` clause, and it always comes last. It defines the items that are included in the result. What more can one say about it?

Usually the XQuery `return` clause generates a single item each time it is evaluated. In general, though, it can produce a sequence. For example, you can do this:

```
for $v in //video[genre="comedy"]
return //actor[@id = $v/actorRef]
```

which selects all the actors for each comedy video. However, the result is a little unsatisfactory, because we cannot tell which actors belong to which video. It is much more common here to construct an element wrapper around each result:

```
for $v in //video[genre="comedy"]
return
  <actors video="{ $v/title }">
    { //actor[@id = $v/actorRef] }
  </actors>
```

We have not discussed XQuery element and attribute constructors until now. But in practice, a FLWOR expression without element constructors can only produce flat lists of values or nodes, and that is not usually enough. We usually want to produce an XML document as the output of the query, and XML documents are not flat.

This means that very often, instead of doing purely relational joins that generate a flat output, we want to construct hierarchic output using a number of *nested* FLWOR expressions. Here is an example that (like the previous query) lists the videos for each actor, but with more structure this time:

```
for $v in //video[genre="comedy"]
return
  <actors video="{ $v/title }">
    { for $a in //actor[@id = $v/actorRef]
      return
        <actor>
          <firstname>{substring-after($a, ",")}</firstname>
          <lastname>{substring-before($a, ",")}</lastname>
        }
    }
  </actors>
```

Here we really do have two nested XQuery loops. The two queries below are superficially similar, and in fact they return the same result:

```
for $i in 1 to 5
for $j in ("a", "b", "c")
return concat($j, $i)
```

and:

```
for $i in 1 to 5
return
  for $j in ("a", "b", "c")
  return concat($j, $i)
```


But now add an `order by` clause to both queries so they become:

```
for $i in 1 to 5
for $j in ("a", "b", "c")
order by $j, $i
return concat($j, $i)
```

and:

```
for $i in 1 to 5
return
  for $j in ("a", "b", "c")
  order by $j, $i
  return concat($j, $i )
```

The difference now becomes apparent. In the first case the result sequence is a1, a2, a3, ... b1, b2, b3,. In the second case it remains a1, b1, c1,... a2, b2, c2. The reason is that the first query is a single FLWOR expression (one `return` clause), and the `order by` clause affects the whole expression. The second query consists of two nested loops, and the `order by` clause can only influence the inner loop.

So, the `return` clause might seem like the least significant part of the FLWOR, but a misplaced `return` can make a big difference in the result. Consider always aligning the F, L, O, W, and R clauses of a single FLWOR expression underneath each other, and indenting any nested expressions, so that you can see what is going on. You can do this easily with the Stylus Studio XQuery Editor.

Other Parts of the XQuery FLWOR Expression

We have explored the five clauses of the FLWOR expression that give it its name. But there are a few details we have not touched on, partly because they are not used very often. They are summarized in this section.

Declaring XQuery Types

In the `for` and `let` clauses, you can (if you wish) declare the types of each variable. Here are some examples.

```
for $i as xs:integer in 1 to 5 return $i*2
```

```
for $v as element(video) in //video return $v/runtime
```

```
let $a as element(actor)* := //actor return string($a)
```

Declaring types can be useful as a way of asserting what you believe the results of the expressions are, and getting an error message (rather than garbage output) if you have made a mistake. It helps other people coming along later to understand what the code is doing, and to avoid introducing errors when they make changes.

Unlike types declared in other contexts such as function signatures (and unlike variables in XSLT 2.0), the types you declare must be exactly right. The system does not make any attempt to convert the actual value of the expression to the type you declare – for example it will not convert an integer to a double, or extract the string value of an attribute node. If you declare the type as string but the expression delivers an attribute node, that is a fatal error.

XQuery Position Variables

If you have used XSLT and XPath, you have probably come across the `position()` function, which enables you to number the items in a sequence, or to test whether the current item is the first or the last. FLWOR expressions do not maintain an implicit context in this way. Instead, you can declare an auxiliary variable to hold the current position, like this:

```
for $v at $pos in //video
  where $pos mod 2 = 0
  return $v
```

This selects all the even-numbered videos – useful if you are arranging the data in a table. You can use `$pos` anywhere where you might use the primary variable `$v`. Its value ranges from 1 to the number of items in the `//video` sequence. If there are no `order by` clauses, then the position variables in each `for` clause follow a nested-loop model as you would

expect. If there is an `order by` clause, the position values represent the position of the items before sorting (which is different from the rule in XSLT).

There are various keywords in the `order by` clause that give you finer control over how the sorting takes place. The most important is the `collation`: unfortunately, though, the way collations work is likely to be very product-dependent. The basic idea is that if you are sorting the index at the back of a book, or the names in a phone directory, then you need to apply rather more intelligent rules than simply sorting on the numeric Unicode code value of each character. Upper-case and lower-case variants of letters may need to be treated the same way, and accents on letters have some quite subtle rules in many languages. The working group defining XQuery settled on the simple rule that every collating sequence you might want has a name (specifically, a URI rather like a namespace URI), and it is up to each vendor to decide what collations to provide and how to name them.

Other things you can say in the order specification include defining whether empty values of the sort key (XQuery's equivalent of null values in SQL) should go at the start or end of the sequence, and whether the sort should be stable, in the sense that items with equal sort key values preserve their original order.

Multiple Assignments

One simple syntax note. Instead of writing

```
for $i in ("a", "b", "c")
for $j in 1 to 5
return concat($i, $j)
```

you can write:

```
for $i in ("a", "b", "c"),
    $j in 1 to 5
return concat($i, $j)
```

The meaning of both is the same. This same technique applies to `let` statements as well.

Grouping

If you are used to SQL, then you might have been wondering what the equivalent to its `DISTINCT` and `GROUP BY` keywords is in XQuery FLOWR expressions. Well, SQL does not have one.

You can, however, get a fair amount of mileage from the `distinct-values()` function. Here is a query that groups videos according to who directed them:

```
<movies>
  {for $d in distinct-values(//director) return
    <director name="{ $d}">
      { for $v in //video[director = $d] return
        <title>{$v/title}</title>
      }
    </director>
  }
</movies>
```

This is not an ideal solution: apart from anything else, it depends heavily on the ability of the query processor to optimize the two nested loops to give good performance. But for the time being, this is all there is. This is an area where vendors are very likely to offer extensions to the language as defined by W3C.

Grouping was a notoriously weak point of XSLT 1.0, and the problem has been addressed with considerable success in the 2.0 version of the language. XQuery will likely follow suit.

Building an XQuery Using the Mapper

This section describes how to build a new XQuery using Stylus Studio's XQuery Mapper.



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the Introduction to the XQuery Mapper video](#).

You can learn more about other video demonstrations of the XQuery Mapper here: http://www.stylusstudio.com/learn_xquery.html#xquery_1.

This section covers the following topics:

- “Process Overview” on page 751
- “Source Documents” on page 752
- “Specifying a Target Structure” on page 757
- “Modifying the Target Structure” on page 760
- “Mapping Source and Target Document Nodes” on page 761
- “Simplifying the Mapper Canvas Display” on page 767
- “Exporting Mappings” on page 768

- [“Searching Document Panes”](#) on page 769
- [“FLWOR Blocks”](#) on page 769
- [“Function Blocks”](#) on page 772
- [“IF Blocks”](#) on page 776
- [“Condition Blocks”](#) on page 776

Process Overview

The process of using the XQuery mapper to build a new XQuery consists of the following steps:

1. Create a new XQuery file in Stylus Studio (**File > New > XQuery File**).
2. Click the **Mapper** tab in the XQuery editor.
3. Add one or more source documents.
4. Specify a target structure.
5. Map source document nodes to target structure nodes. As part of this step, you can optionally define function, FLWOR (For each, Let, Where, Order by, Return), If, and condition blocks to perform actions on source document nodes and map the return value to the target structure node.

Stylus Studio uses the information expressed on the **Mapper** tab to compose an XQuery that returns as its result an XML document that conforms to the structure represented by the target structure you specify.

Each of these steps is described in greater detail in the following sections.

Working with Existing XQueries

You can, of course, open an existing XQuery in Stylus Studio. When you do, the **XQuery Source** page displays the XML used to compose the XQuery, and the **Mapper** tab displays the source documents, target structure, and source-target mappings that can be inferred from the source XQuery file. All of the procedures described in this section can be performed on new or existing XQuery documents.

Saving the Mapping

You can save the XQuery mapping – source and target document trees, as well as the contents of the Mapper canvas – as an image. See [“Exporting Mappings”](#) on page 768.

Source Documents

In Stylus Studio, a source document can be an XML document, an XML Schema (XSD), or a document type definition (DTD). The role of a source document is to provide Stylus Studio with a structure that it can use to compose the XQuery, based on how you map individual source document elements and attributes to nodes in the target structure. Stylus Studio infers the structure from the document you specify and displays this structure on the **Mapper** tab.

This section covers the following topics:

- [“Choosing Source Documents”](#) on page 752
- [“Source Documents and XML Instances”](#) on page 752
- [“How to Add a Source Document”](#) on page 755
- [“How to Remove a Source Document”](#) on page 756
- [“How Source Documents are Displayed”](#) on page 756

Choosing Source Documents

You can use one or more source documents to build an XQuery in Stylus Studio. You might want to select multiple documents if you need their elements or attributes to fully describe the target structure or the desired XQuery result content, for example.

If you choose an XSD or DTD document, you must also choose an XML instance document to associate with it. Stylus Studio uses the instance document associated with a XSD or DTD source document to generate the XPath document() function in the finished XQuery code. This document is also used to preview XQuery results.

See [“Source Documents and XML Instances”](#) on page 752 to learn more about how Stylus Studio treats source documents. See [“Creating an XQuery Scenario”](#) on page 800 to learn more about XQuery scenarios.

Source Documents and XML Instances

As described previously, Stylus Studio uses the source document you specify to infer a structure you can use to create mappings to the target structure. In addition to the document structure, Stylus Studio needs document content information in order to compose a complete XQuery. You provide this information by associating a XML instance to each source document you specify.

Source documents can have one of three associations, each of which has implications for the XPath expressions written by Stylus Studio when it composes the XQuery code. A source document can be associated with

- Itself. That is, the document represented by structure displayed on the **Mapper** tab and the XML instance are one in the same. In this situation, Stylus Studio generates the document() function in the XQuery code. For example:

```
for $book in document("file:///c:\Program Files\Stylus Studio\examples\simpleMappings\catalog.xml")/books/book
```

- The XML document specified in the optional XQuery scenario. Only one source document can be associated with the XQuery scenario. In this situation, Stylus Studio does not generate the document() function in the XQuery code. For example:

```
for $book in /books/book
```

The document() function is not necessary because Stylus Studio uses the XQuery input document specified in the **Scenario Properties** dialog box.

By default, Stylus Studio uses the first XML document you add to the XQuery mapper as the source XML for the XQuery scenario, as shown here:

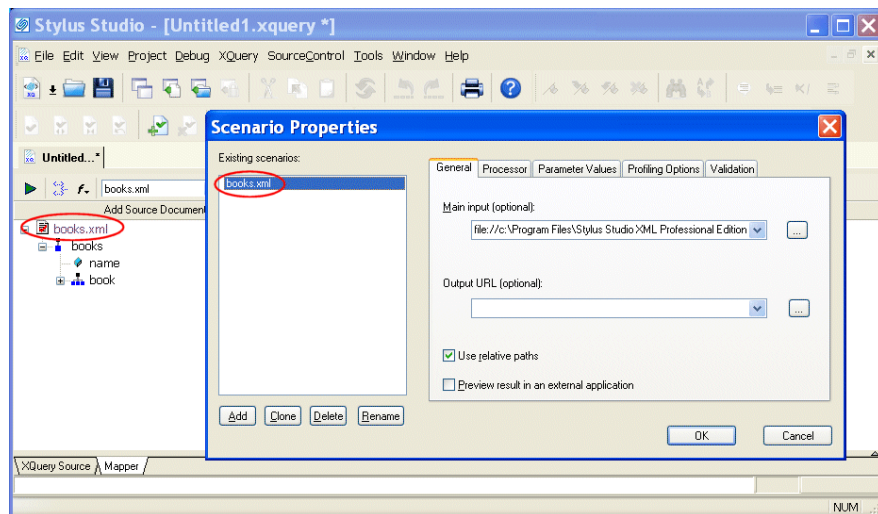


Figure 294. Default Source Document

The document specified in the **Source XML URL** field on the **Scenario Properties** dialog box is the document used to preview XQuery results. You can select this

association for another XML document if you choose, but only one source document may have this association.




Note Creating a scenario for an XQuery is optional. See [“Creating an XQuery Scenario”](#) on page 800.

- Some other XML instance. A XSD or DTD document used as an XQuery source document must always be associated with an XML instance. In this situation, Stylus Studio generates the `document()` function in the XQuery code.

Source document icons

Stylus Studio uses different icons to indicate how a source document is associated with the other documents used to compose the XQuery.

Table 94. Source Document Icons

<i>Icon</i>	<i>Meaning</i>
	The source document is associated with itself. This is the default for most XML documents (and XML documents only).
	The source document is associated with the XML document specified in the XQuery scenario. This is the case with the first XML document you add to XQuery mapper, but you can change this association manually if you choose. See “How to Change a Source Document Association” on page 754.
	The source document is associated with a separate XML document instance. XSD and DTD source documents are always associated with an XML instance.

How to Change a Source Document Association

◆ **To change a source document association:**

1. Right click the source document whose association you want to change. The source document shortcut menu appears.
2. Click **Associate With**, and then select the document you want to associate with the source document.

How to Add a Source Document

◆ To add an XQuery source document to XQuery mapper:

1. Click the **Mapper** tab if necessary.
2. Click the **Add Source Document** button at the top left of the **Mapper** tab.
The **Open** dialog box appears.
3. Select the document you want to use as the source document for building the XQuery.
4. Click **Open**.

If you selected an XML document in [Step 3](#), the document appears in the source document pane of the **Mapper** tab. Go to [Step 5](#).

If you selected an XSD or DTD document, Stylus Studio displays the **Choose Root Element** dialog box.

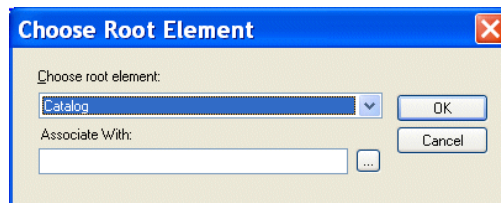



Figure 295. Choose Root Element Dialog Box

You use this dialog box to associate the XSD or DTD with an XML instance.

Note

The **Associate With** field appears only when you add a second document to the XQuery mapper source and that document is an XSD or DTD. You use it to specify the XML instance that you want to associate with the XSD or DTD. This field does not appear if the XSD or DTD is the first source document you add to the XQuery mapper – Stylus Studio uses the XML Source document specified in the **Scenario Properties** dialog box as the XML instance in this case.

- a. Select the element from the XSD or DTD document that you want to use as the root element. The **Choose root element** drop-down list displays elements defined in the document you selected in [Step 3](#).
- b. Use the **Browse** () button to specify the XML instance to which you want to map the root element you have selected. The root element of the XML document you select must be the same as the element you selected as the root element from the XSD or DTD document.

- c. Click **OK**.
The document appears in the source document pane of the **Mapper** tab. Go to [Step 5](#).
5. To add another source document, return to [Step 2](#).

How to Remove a Source Document

Note A source document cannot be removed from XQuery mapper if it is mapped to the target structure. See [“Removing Source-Target Map”](#) on page 766.

◆ **To remove a source document from XQuery:**

1. Remove any maps from the source document to the target schema. (See [“Removing Source-Target Map”](#) on page 766 if you need help with this step.)
2. Right click on the source document.
The source document shortcut menu appears.
3. Select **Remove Schema**.

How Source Documents are Displayed

A source document is represented using a page icon, and its name is displayed using a different color to help distinguish it from element and attribute names. The page icon is modified based on the source document’s association with other documents. See [“Source Documents and XML Instances”](#) on page 752 for more information on this topic.

By default, only the file name itself is displayed; if you want, you can display the document’s full path by selecting **Show Full Path** on the document’s shortcut menu. (Right-click on the document name to display the shortcut menu.)

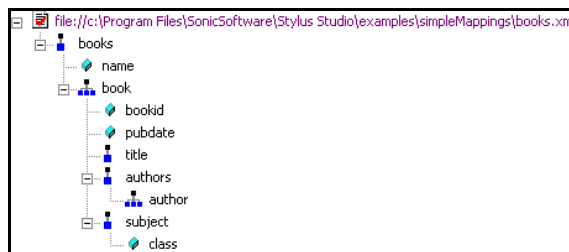





Figure 296. Source Document Display

Source documents are displayed using the tree view; you can use your keyboard's *, +, and - number pad keys to expand and collapse selected documents.

Document structure symbols

Stylus Studio uses the following symbols to represent nodes in both source and target document structures:

Table 95.

<i>Symbol</i>	<i>Meaning</i>
	Repeating element
	Element
	Attribute

See “[Source document icons](#)” on page 754 to learn about the different ways source document icons are depicted.

Tip If a node is required by the XML Schema or DTD associated with the target document, a red check appears over the node symbol.

Getting source document details

If you want details about the document that are not available in tree view, you can open the document by selecting **Open** from the document's shortcut menu. When you open a document this way, Stylus Studio displays it in its own editor (the XML editor if it is an XML document, for example).

Specifying a Target Structure

There are two ways to specify an XQuery target structure:

- You can select an existing document from which Stylus Studio infers a structure and, optionally, modify the structure. Existing nodes in a target structure are displayed in blue. Nodes that you add are displayed in red. If a node is required by the associated XML Schema or DTD, a red check appears over the node symbol.
- You can build a structure from scratch, starting with the root element and defining other elements and attributes as needed. Nodes for target structures you define are displayed in red.

This section covers the following topics:

- [“Using an Existing Document”](#) on page 758
- [“Building a Target Structure”](#) on page 758

See [“Modifying the Target Structure”](#) on page 760 to learn about the types of changes you can make to a target structure.

Using an Existing Document

◆ **To use an existing document to provide the XQuery target structure:**

1. Click the **Mapper** tab if necessary.
2. Click the **Set Target Document** button at the top left of the **Mapper** tab.
The **Open** dialog box appears.
3. Select the document you want to use to provide the target structure for defining the XQuery.
4. Click **Open**.
The structure of the document you select appears in the target document pane of the **Mapper** tab.

Building a Target Structure

To build a target structure from scratch, you first create a root element, and then define child elements and attributes as needed.

How to create a root element

◆ **To create a root element:**

1. Click the **Mapper** tab if necessary.
2. Right click the area underneath the **Set Target Document** button.
The target document shortcut menu appears.

3. Select **Create Root Element**.
The **Name** dialog box appears.



Figure 297. Name Dialog Box

4. Type a name for the root element and click **OK**.
The root element you specified appears in the target document pane of the **Mapper** tab.

Note You can create elements and attributes in a new or existing target structure.

How to create elements and attributes

◆ **To create elements and attributes:**

1. Click the **Mapper** tab if necessary.
2. Select the attribute or element to which you want to add a child element or attribute.
If you have just created a root element, select the root element.
3. Right click the area underneath the **Set Target Document** button.
The target document shortcut menu appears.
4. Choose one of the following:
 - **Add Attribute**
 - **Add Child Element**
 - **Insert Element After** (This choice is not applicable to the root element; it creates the element as a sibling of the selected element.)

The **Name** dialog box appears.



Figure 298. Name Dialog Box

5. Type a name for the node and click **OK**.
The node you specified is added to the target structure in the **Mapper** tab.

Modifying the Target Structure

This section describes the techniques you can use to modify the structure and content of an XQuery mapper target structure. It covers the following topics:

- “[Adding a Node](#)” on page 760
- “[Removing a Node](#)” on page 760
- “[Setting a Text Value](#)” on page 760

Adding a Node

See “[How to create elements and attributes](#)” on page 759.

Removing a Node

Note Before you can remove a node, you must delete any links to that node. See “[Removing Source-Target Map](#)” on page 766.

◆ **To remove a node from the target structure:**

1. Remove any links to the node you want to remove from the target structure. See “[Removing Source-Target Map](#)” on page 766 if you need help with this step.
2. Select the node and press the Delete key.

Alternative: Right-click the node and select **Remove Node** from the shortcut menu.

Setting a Text Value

You can set text values for target structure elements and attributes. You might want to do this if you are composing an XQuery with an element or attribute that requires a fixed value, instead of using a value gathered from an input XML document.

Here is the XQuery code Stylus Studio generates for the `Title` element when a text value is specified for it:

```
<Book>
  <Title>Confederacy of Dunces</Title>
</Book>
```

Stylus Studio displays a red letter **T** for nodes for which you define a text value: Title .

◆ **To set a text value for a target structure node:**

1. Right-click the node for which you want to set the text value.
The shortcut menu appears.
2. Select **Set Text Value** from the shortcut menu.
The **Value** dialog box appears.



Figure 299. Value Dialog Box

3. Type the string you want to use as the text value and click **OK**.

Mapping Source and Target Document Nodes


You map a source document node to a target structure node using drag and drop to create a link between the two nodes. Stylus Studio composes XQuery code based on these maps.

This section covers the following topics:

- “Preserving Mapper Layout” on page 761
- “Left and Right Mouse Buttons Explained” on page 762
- “How to Map Nodes” on page 763
- “Link Lines Explained” on page 763
- “Removing Source-Target Map” on page 766

Preserving Mapper Layout

As you add function blocks to the XQuery mapper, Stylus Studio places them in the center of the mapper canvas. You can change the default placement of function blocks by dragging and drag and dropping them where you like. Stylus Studio preserves the placement you select within and across sessions (as you toggle between the mapper and the **XQuery Source** tab, for example).

As you use the splitter in the XQuery mapper to widen the source and target document panes, the size of the mapper canvas is reduced. The **Fit in Mapper Canvas** button ()

located at the top of the XQuery mapper, redraws the diagram in whatever space is currently available to the mapper canvas. This feature is also available from the mapper short-cut menu (right-click anywhere on the mapper canvas to display the short-cut menu).

Tip You can also show links for visible nodes, or links for just the node you select. See [“Simplifying the Mapper Canvas Display”](#) on page 767.

Left and Right Mouse Buttons Explained

You can use either the left or the right mouse button to perform the drag and drop operation used to create source-target mappings in XQuery.

If you use the left mouse button to perform the drag operation, the link always maps the source node to the target node, one-to-one, without making any changes to the target structure.

If you use the right mouse button, Stylus Studio displays a shortcut menu that provides you with alternatives for modifying the target structure.

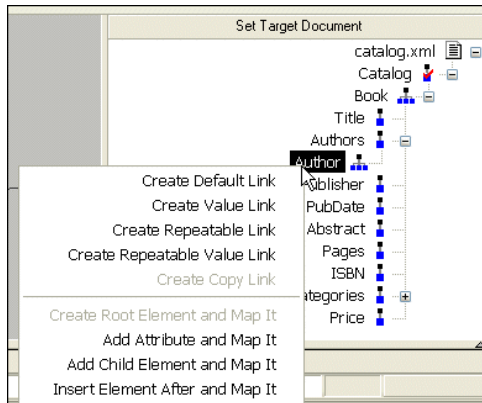


Figure 300. Linking Using the Right Mouse Button Displays a Shortcut Menu

Using this menu, you can easily perform many operations. For example, you can

- Map a source document node to an existing target structure node – this menu choice, **Map to This Node**, is the same as creating the link using the left mouse button.
- Add a source document node (element or attribute) as an attribute of the target structure node you select and map the two nodes.
- Add a source document node as a child element of the target structure node you select and map the two nodes.

- Add a source document node as a sibling of the target structure node you select and map the two nodes.
- Copy the entire source document node – its structure and its content – to the target structure and map it.

How to Map Nodes

◆ To map nodes:

1. Using either the left or right mouse button, drag the source document element or attribute to the appropriate node on the target structure.
2. When the pointer is on the appropriate target element, release the mouse button to complete the link.

Link Lines Explained

Stylus Studio draws lines for the maps you create from source document nodes to target structure nodes. Different line styles are used to convey information about the XQuery represented by the node mapping. There are three line styles:

- Thin
- Dashed
- Thick

The sample files used to illustrate these styles are `books.xml` and `catalog.xml`, from the Stylus Studio `examples\simpleMappings` directory.

Thin line

A thin line indicates that the XQuery code generated by Stylus Studio copies content from the source node to the target node. Such a line is created when you map one element or attribute to another using the left mouse button, or any of the following choices on the map shortcut menu:

- **Create Root Element and Map It**
- **Add Attribute and Map It**
- **Add Child Element and Map It**
- **Insert Element After and Map It**

In addition, the structure required to navigate to the node is also generated if it does not already exist in the XQuery. For example, consider the map between the `title` element in `books.xml` and the `Title` element in `catalog.xml`:

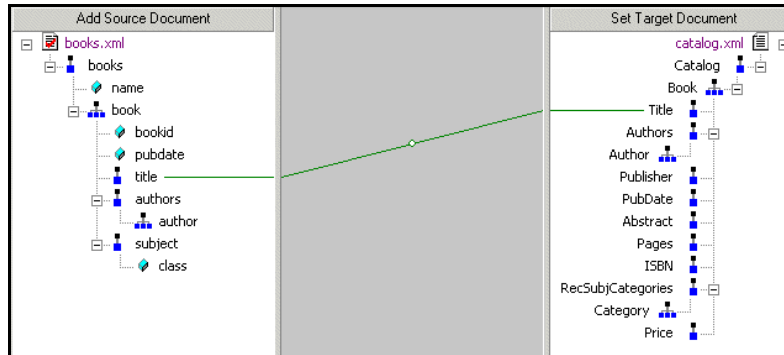


Figure 301. Thin Lines in XQuery Mapper

This map results in Stylus Studio composing the following XQuery code:

```
<Catalog>
  <Book>
    <Title>
      {/books/book/title/text()}
    </Title>
  </Book>
</Catalog>
```

The content is expressed as `{/books/book/title/text()}`, and this statement is preceded by the structure needed to locate the `title` element content.

Dashed line

A dashed line indicates that only structure code is being generated. Such a line is created when you use a FLWOR or IF block. For example, consider the map between the book and Book repeating elements:

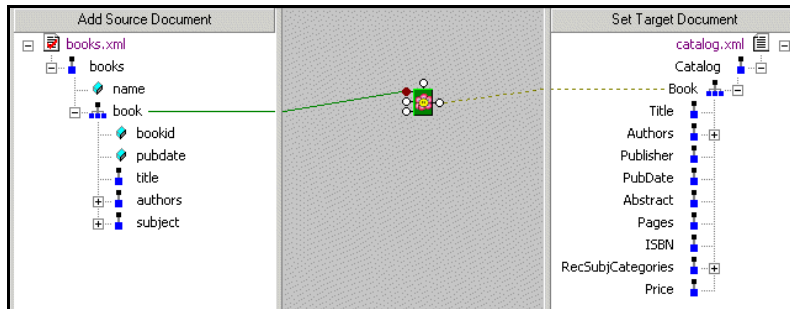


Figure 302. Dashed Lines in XQuery Mapper

A map involving a FLWOR block results in the following code:

```
<Catalog>
{
  for $book in /books/book
  return
  <Book>
    <Title/>
  </Book>
}
</Catalog>
```

Notice that the FOR loop returns only structure (shown in *italics*), not content. To add content, we could also map the `title` element to the `Title` element, which results in the following:

```
<Catalog>
{
  for $book in /books/book
  return
  <Book>
    <Title>
      { $book/title/text() }
    </Title>
  </Book>
}
</Catalog>
```

Of course, the FLWOR block can be used to define much more complex expressions, involving maps from source document nodes to its WHERE and ORDER BY ports, for example.

Thick line

A thick line indicates that the XQuery code generated by Stylus Studio replicates the complete structure and content of the source document node in the target. Such a map is created when you use the **Copy Node** choice on the link shortcut menu. Consider the following map – the `bookid` attribute on the source was copied to the target as a child of the `Book` repeating element:

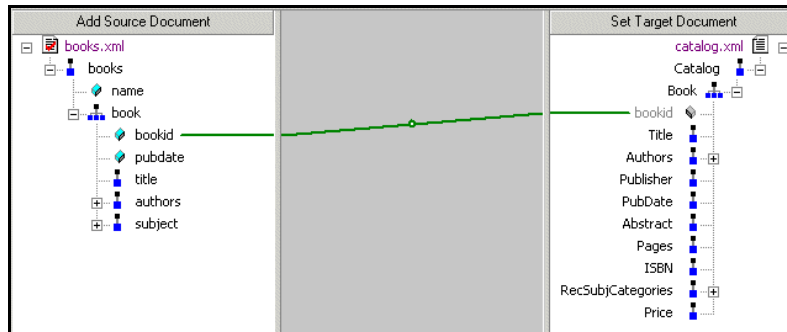


Figure 303. Thick Lines in XQuery Mapper

For this type of map, Stylus Studio creates the XQuery code required to duplicate the source structure and content in the target, as shown in the following sample:

```
<Catalog>
  <Book>
    {/books/book/@bookid}</Book>
</Catalog>
```

Notice that the `bookid` attribute is displayed in gray in the target structure pane. This indicates that you cannot edit it.

Removing Source-Target Map

◆ **To remove a map from a source document node to a target element node:**

1. Select the line that represents the map you want to delete.

Note Select the portion of the line that is drawn on the XQuery mapper canvas.

2. Press the **Delete** key.

Alternative: Select **Delete** from the line shortcut menu (right click on the line to display the shortcut menu).

Simplifying the Mapper Canvas Display

By default, the XQuery Mapper displays all links between source and target document nodes, regardless of whether or not the node associated with a link is currently visible in the **Source Document** or **Target Document** pane. Further, as your XQuery code becomes more complex, the mapper canvas can become dense with graphical representations of the functions defined in the code and the links that represent them. Consider this example of XML-Q4.xquery, one of the sample XQuery documents in the Examples project installed with Stylus Studio.

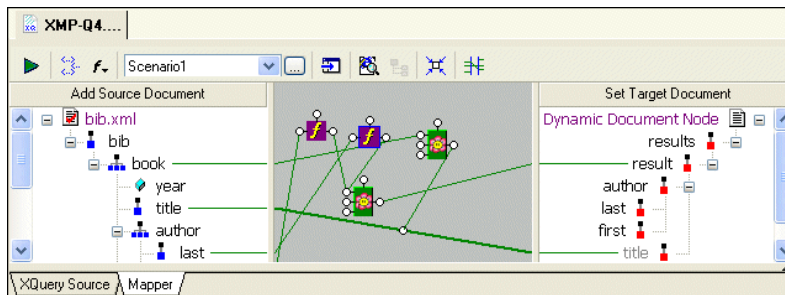


Figure 304. Mapper Shows Links to All Nodes, Visible or Not

You can hide links for nodes that are not currently visible in the **Source Document** or **Target Document** pane by clicking the **Hide Links for Nodes that are not Visible** button, as shown in [Figure 305](#):

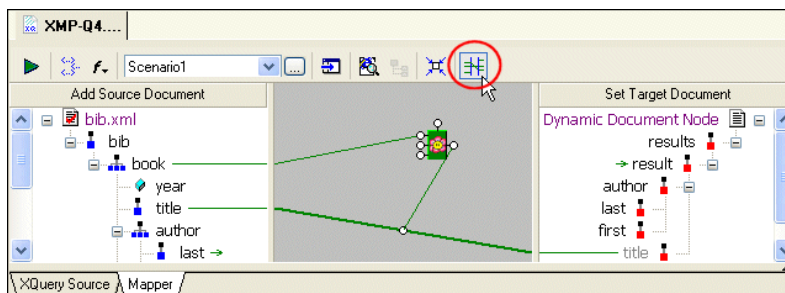


Figure 305. Simply the Mapper by Hiding Links

When you use this feature, Stylus Studio displays

- Links in the Mapper canvas only if both nodes are currently visible in the document panes

- Green arrows (like the ones shown in [Figure 306](#)) in the document panes if only one of two linked nodes is currently visible.

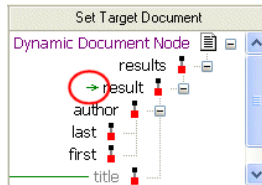


Figure 306. Arrows Identify Partially Available Links

Other Mapper Display Features

In addition to displaying links for only those nodes that are visible in both document panes, you can use the document node shortcut menu (right-click on a node in a document pane) to

- Show links to a specific node
- Hide links to a specific node
- Show/hide all links

Exporting Mappings

You can export a mapping – source and target document trees and Mapper canvas contents – as an image file. The default image format is JPEG (.jpg), but you can choose from other popular image file formats such as .bmp and .tiff.

The exported image reflects the document trees at the time you export the image – if you have collapsed a node in Stylus Studio, for example, that node is also collapsed in the exported image. However, the exported image includes the entire document tree and Mapper canvas, not just what is currently visible on the **Mapper** tab.

By default, all source-target document links are displayed. However, if you have chosen to [hide or show links](#) for only certain nodes, the exported image reflects that choice and displays only the links for the nodes as you have specified. See [“Simplifying the Mapper Canvas Display”](#) on page 767 for more information on hiding and showing links.

◆ **To export an XQuery mapping:**

1. Optionally, hide links for any nodes in the source or target documents that you do not want to appear in the exported image.
2. Select **XQuery > Export Mapping as Image** from the Stylus Studio menu. Stylus Studio displays the **Save As** dialog box.
3. Specify a URL for the file.
4. Optionally, change the image type. (The default is JPEG; .bmp and .tiff are also available.)
5. Click **Save**.

Searching Document Panes

You can search document panes using the **Find** dialog box.

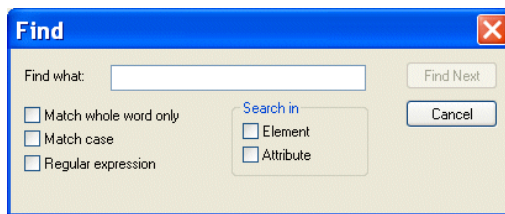


Figure 307. You Can Search Document Panes

You can restrict your search to elements and/or attributes, and you can even search using regular expressions to define your match pattern.

◆ **To display the Find dialog box:**

1. Right-click in the document pane.
2. Select **Find** from the shortcut menu.

FLWOR Blocks

This section describes how to work with FLWOR blocks in the XQuery **Mapper** tab. It covers the following topics:

- [“Parts of a FLWOR Block”](#) on page 770
- [“Creating a FLWOR Block”](#) on page 771

Parts of a FLWOR Block

FLWOR blocks are drawn as a green block with an illustration of a flower at its center, and five connectors, called *ports*, placed along the block's border:

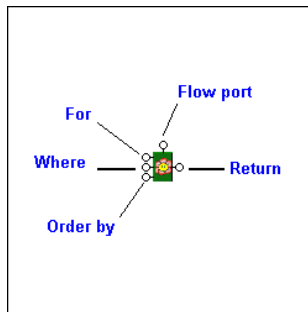


Figure 308. FLWOR Block

For, Order by, and Return ports

You define a FLWOR statement's **For** and **Order by** clauses by mapping source document elements and attributes to them, as appropriate. For example, if you wanted your XQuery to return a list of books ordered by publication date, you would map the book repeating element in `books.xml` to the FLWOR block's **For** port, and the **Return** port to the `Book` repeating element in `catalog.xml`. (As an alternative, you could map the two repeating elements directly, and Stylus Studio would create the FLWOR block and this mapping for you automatically, as described in [“Creating a FLWOR Block”](#) on page 771). Next, you would map the source document `pubdate` attribute to the **Order by** port. For a FLWOR block defined in this way, Stylus Studio generates the following XQuery:

```
<Catalog>
  {
    for $book in /books/book
    order by $book/@pubdate
    return
    <Book/>
  }
</Catalog>
```

Where port

The input for the **Where** port must be the output port of another block, such as a condition, IF, or function block. Imagine you have two source documents – you can create an Equal condition block, and specify that the content of an element in one source document must match the content of an element in the other source document, and map the return value

of this condition to the **Where** port on the FLWOR block. Creating an Equal condition that specifies that the `bookid` attribute must be equal to the `title` element results in Stylus Studio generating the following XQuery code, for example:

```
<Catalog>
  {
    for $book in /books/book
    where $book/@bookid = $book/title
    order by $book/@pubdate
    return
    <Book/>
  }
</Catalog>
```

See “[IF Blocks](#)” on page 776 and “[Function Blocks](#)” on page 772 for information on using other types of blocks in XQuery mapper.

Flow port

The **Flow** port, which is also present on IF and function blocks, allows you to link the result from other FLWOR, IF, and function blocks to define a conditional execution order for your XQuery expressions. You might decide you want a particular **For** each statement executed only after performing a certain function, for example. Inputs for the **Flow** port include the **Return** port of IF, function, and other FLWOR blocks.

Creating a FLWOR Block

You can create FLWOR blocks in the XQuery **Mapper** tab in one of two ways:

- Right-click on the mapper canvas and select **New | FLWOR Block** from the shortcut menu.
- Map one repeating element to another – Stylus Studio automatically creates a FLWOR block, mapping the source document node to the **For** port, and the **Return** port to the target structure node. Consider this code, which Stylus Studio generated after mapping the `book` repeating element in `books.xml` to the `Book` repeating element in `catalog.xml`:

```
<Catalog>
  {
    for $book in /books/book
    return
    <Book/>
  }
</Catalog>
```

Function Blocks

Stylus Studio supports standard functions defined by the W3C and any user-defined functions you might have created. This section describes how to work with function blocks in Stylus Studio and covers the following topics:

- [“Creating a Function Block”](#) on page 772
- [“Parts of a Function Block”](#) on page 773
- [“User-Defined Functions”](#) on page 774
- [“concat Function Blocks”](#) on page 775

Standard Function Block Types

Stylus Studio provides graphic support for the following types of XQuery functions:

- Accessor
- Aggregate
- Boolean
- Date time
- Error
- Node
- Notation
- Number
- QName
- Sequence
- Sequence generator
- String

If a standard function does not provide the functionality you need, create a user-defined function. See [“User-Defined Functions”](#) on page 774.

Creating a Function Block

The procedure for creating standard and user-defined function blocks varies slightly:

- ◆ **To create a standard function block:**
 1. Right-click on the mapper canvas.

2. Select **New > Function Block** from the shortcut menu. Available functions are displayed in submenu categories.

◆ **To create a user-defined function block:**

1. Right-click on the mapper canvas.
2. Select **New > User Functions** from the shortcut menu.

Any user-defined functions defined in the XQuery source are displayed in a sublist. See [“User-Defined Functions”](#) on page 774 to learn more about creating user-defined functions in Stylus Studio.

Parts of a Function Block

Function blocks are drawn as a purple block with an italic “f” at its center, and connectors, called *ports*, placed along the block’s border. Input ports (none or more based on the function), the **Flow** port at the top, and the **Return** port on the right:

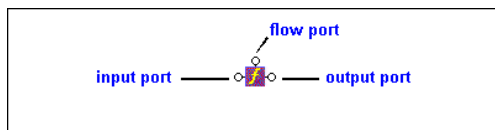


Figure 309. Function Block

Input ports

Input ports are on the left side of the function block. The number and definition of input ports varies from function to function. To specify a value for an input port, drag a source document element or attribute to the port and release it.

Flow port

Flow ports, on the top of function blocks, are the same for FLWOR, function, and IF blocks. See [“Flow port”](#) on page 771.

Return port

The **Return** port is on the right side of the function block. You use the **Return** port to map the function result directly to a target structure element or attribute, or to a FLWOR, IF, condition, or another function block.

User-Defined Functions

You can declare your own functions in XQuery, such as the following:

```
declare function total-price( $i as element) as xs:decimal
{
  let $subtotals := for $s in $i return $s/quantity * $s/USPrice
  return sum( $subtotals )
}
```

This particular user-defined function, which takes an element as its argument and returns a sum of the prices, might be used as follows (shown in italics):

```
<Catalog>
{
  for $book in /books/book
  return
  <Book>
    <Price>
      {total-price($book/@bookid)}
    </Price>
  </Book>
}
</Catalog>
```

When you create a user-defined function, Stylus Studio adds it to the **New > User Functions** shortcut menu available when you right-click the mapper canvas.

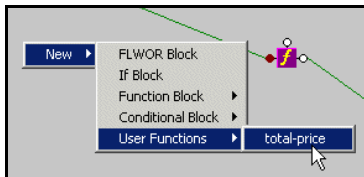


Figure 310. User-Defined Functions

This makes it easy to reuse a user-defined function on the **Mapper** tab once it has been defined in the XQuery source.

concat Function Blocks

There are three types of concatenation (`concat`) functions for strings:

- `concat()` as `string` allows you to specify a literal value that you might wish to concatenate to some other value in your XQuery.



Figure 311. `concat()` as `string`

- `concat($op1 as string?)` as `string` allows you to specify a variable that you might wish to concatenate to some other value.

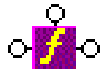


Figure 312. `concatn($op1 as string?)`

- `concat($op1 as string?, $op2 as string?, ...)` as `string` allows you to concatenate two or more variables.



Figure 313. `concat($op1 as string?, $op2 as string?, ...)`

Note that only the first two input ports are associated with variables (`$op1 as string?` and `$op2 as string?`). When you map a value to the third input port (`...`), Stylus Studio automatically adds a fourth input port to allow you concatenate a fourth value. This behavior is repeated for each additional string you define.

IF Blocks

IF blocks have a single input port, labeled **condition**; a **Flow** port; and two result ports: **if then**, and **if else**.

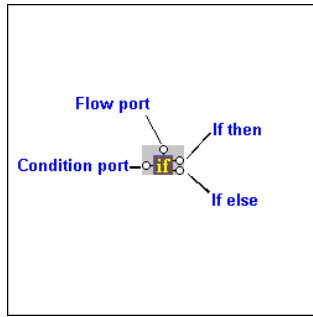


Figure 314. IF Block

You use IF blocks to compose `if then, else` XQuery expressions, such as the following:

```
<Book>
  {
    if( $book/title ) then
      <Title/>
    else
      <ISBN/>
  }
</Book>
```

This expression, for example, was composed by mapping

- The `title` element in the source document to the IF block's input port.
- The **if then** result port to the `title` element in the target structure.
- The **if else** result port to the `ISBN` element in the target structure.

IF blocks create a structure if the `if then` or `if else` branches are true. These ports can be connected to the target schema; otherwise they can be connected to **Flow** ports of FLWOR, function, and other IF blocks.

Condition Blocks

The Stylus Studio XQuery mapper allows you to graphically define the following types of conditions:

- Equal (=)
- Less than (<)

- Greater than (>)
- Less than or equal to (<=)
- Greater than or equal to (>=)
- and (&)
- or (||)

All condition blocks have two input ports and a single **Return** port, as shown in this example of a greater than block.



Figure 315. Greater Than Block

You can map the **Return** port to a target structure element or attribute, or to the input port on a FLWOR, function, IF, or another condition block.

Working with the XQuery collection() Function



Support for the XQuery `collection()` function is available only in Stylus Studio XML Enterprise Suite.

As implemented in Stylus Studio, the XQuery `collection()` function allows you to include relational database tables and views in your XQuery as if they were XML documents. (The `collection()` function is implementation-specific – different vendors have implemented it in different ways. In some implementations, for example, the `collection()` function takes as its argument a URL that specifies an XML document.)

This section describes how to work with the `collection()` function in Stylus Studio.



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the XQuery Collections video](#).

A complete list of the videos demonstrating Stylus Studio's features is here: http://www.stylusstudio.com/xml_videos.html.

This section covers the following topics:

- “Using the `collection()` Function in Stylus Studio” on page 778
- “How the `collection()` Function is Processed” on page 778
- “Creating a Database Connection” on page 779
- “Creating a `collection()` Statement” on page 782

- [“Other Ways to Register a Database Configuration”](#) on page 785

Using the `collection()` Function in Stylus Studio

The process of using the `collection()` function in Stylus Studio consists of these basic steps:

1. Create a connection to the database server whose tables and/or views you want to query. You create database connections from the **File Explorer** window.
2. Register the database connection with your XQuery document. This process allows the database’s tables and views to be used in your XQuery code.
3. Invoke the `collection()` function in your XQuery code. You can type the `collection()` statement by hand, or have Stylus Studio create it for you.
4. Ensure that the processor specified in the XQuery scenario is either the Stylus Studio built-in processor or the DataDirect XQuery[®] processor.

These steps are described in greater detail later in this section.

How the `collection()` Function is Processed

The Stylus Studio built-in and DataDirect XQuery[®] processors process `collection()` functions differently:

- The DataDirect XQuery[®] processor converts the XQuery code to SQL statements and pushes the SQL directly to the database server. Results are returned to Stylus Studio as XML and displayed in the **Preview** window.
- Stylus Studio's built-in processor retrieves the entire table or view, converts the data into XML, and evaluates the query against the XML.

Because it processes XQuery as SQL on the database server, the DataDirect XQuery[®] processor can provide performance superior to that of other XQuery processors when querying relational data as XML.

Database Connections

The database connection is established when the XQuery code is executed and closed as soon as a result is returned. Connection settings used are those associated with the data source used to create the XQuery. See [“Creating a Database Connection”](#) on page 779 for more information on this topic.

Handling Invalid Characters

Some characters, like spaces, are valid in SQL but are invalid in XML. Invalid characters are escaped using SQL/XML escaping convention when the relational data is converted to XML. For example, Stylus Studio would create an XML tag for a column named `last name` as `last_x0020_name`.

Creating a Database Connection

Before you can execute a `collection()` function in an XQuery document, you need to create a database connection. This is part of the process of making the database tables and views available to your XQuery code.

Specific properties vary from database to database, but to connect to a database you typically need to specify

- The *type of database* to which you want to connect. You can connect to one of the default relational databases supported by Stylus Studio (Oracle, Microsoft SQL Server, Informix, and so on).
- The *server URL* and *other connection parameters*. In addition to the server's location, connection parameters can include the server name, the port through which the connection is established, and other information, such as a server ID (SID).
- The *username* and *password* required to log in to the database.

This section describes these connection setting properties in greater detail and.

This section covers the following topics:

- [“Choosing a Database”](#) on page 779
- [“Using the Server URL Field”](#) on page 780
- [“Username and Password”](#) on page 780
- [“How to Create a Database Connection”](#) on page 780

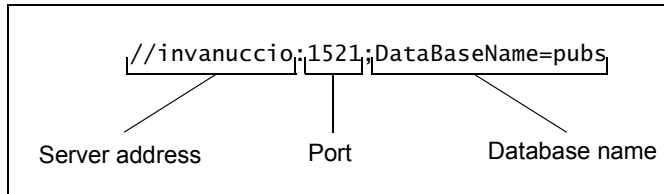
Choosing a Database

Stylus Studio allows you to connect to any of a number of natively supported relational databases, such as Oracle, Microsoft SQL Server, Sybase, Informix, and IBM DB2.

If you want to connect to one of the relational databases supported by Stylus Studio, simply select the database from the **Database Type** drop-down list.

Using the Server URL Field

You use the **Server URL** field to identify the server hosting the database to which you want to connect, the port to use, and any other required or optional parameters. For example, the string used to connect to a Microsoft SQL Server database might look like this:



The specific syntax of the string you enter in the **Server URL** field varies based on database type. Consult your database documentation for information regarding connectivity syntax and optional parameters.

Tip Stylus Studio populates the **Server URL** field with a default string appropriate for the database you specify in the **Type** field.

Username and Password

You use the **Username** and **Password** fields to specify the database user you want to associate with this data source.

How to Create a Database Connection

◆ **To create a database connection:**

1. Display the **File Explorer** window if it is not already open (**View > File Explorer**).
2. In the **File Explorer** window, right-click the **RelationalDB** icon and select **New Server** from the short-cut menu.

Stylus Studio displays the **Connection Settings** dialog box.

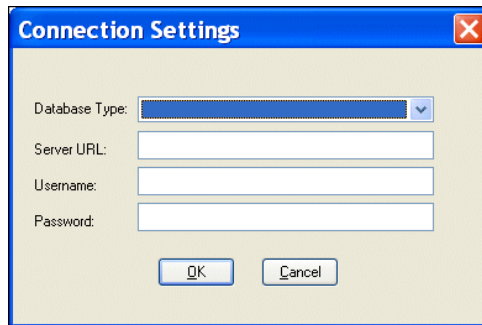


Figure 316. Connection Settings Dialog Box

3. Specify the information needed to create the database connection and click OK.
The server connection appears in the **File Explorer** window.

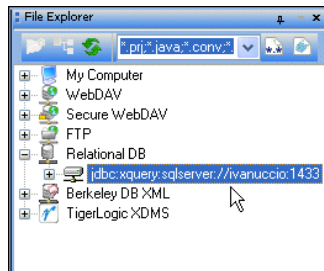


Figure 317. New Database Server Connection

Creating a collection() Statement

This topic describes how to create a `collection()` statement in your XQuery code automatically. If you prefer, you can always write the `collection()` statement manually. Regardless of how you input the `collection()` statement in your XQuery code, Stylus Studio will be able to execute it only if you have created a database connection for the database associated with the table or view referenced in the `collection()` function calls and have registered that connection with the XQuery document.

collection() Function Syntax

The `collection()` function takes as its argument a URI that identifies a specific database table or view, such as this function referencing the `title` column of the `books` table in the `pubs2` database:

```
collection("pubs2.dbo.books")/books/title
```

You can always edit `collection()` functions created by Stylus Studio. As long as they refer to an object that is available based on the database connection associated with the XQuery document, the `collection()` function will execute successfully. See [“Choosing a Database Object”](#) on page 786 for more information on this topic.

What Happens When You Create a collection() Statement?

You create a `collection()` statement by selecting the table or view you want to query from the **File Explorer** window, and dropping it on the editing pane of the **XQuery Source** tab in the XQuery Editor. When you drop the table or view on the editing pane, Stylus Studio

- Automatically creates the `collection()` statement in the XQuery code based on the table or view you selected

- Registers with the XQuery document the connection information for the database associated with the selected table or view, and displays the database in the schema pane of the XQuery Editor, as shown here:

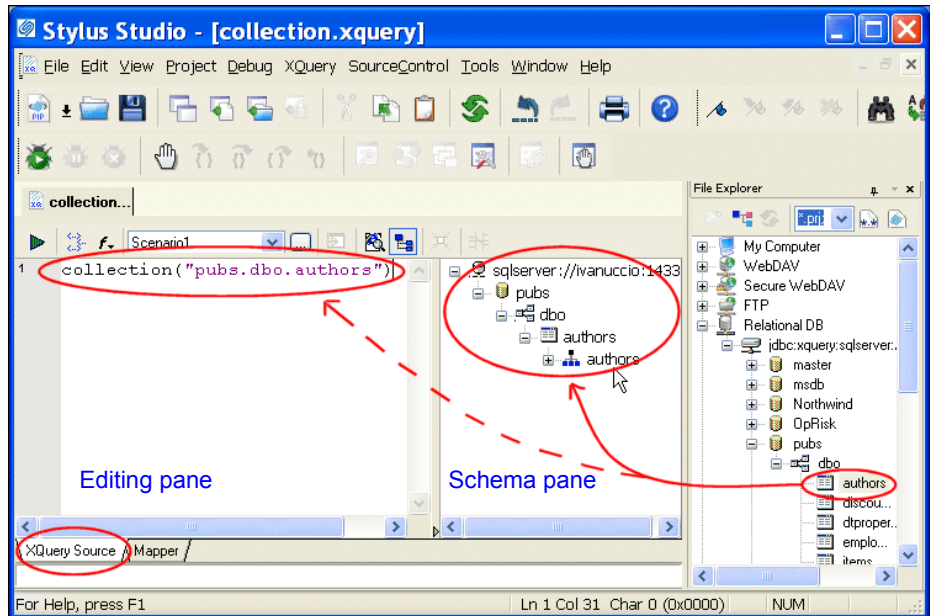


Figure 318. collection() Statements are Created Automatically

Once the database connection information is registered with the XQuery document, you can expand the database nodes in the schema pane to display individual table and view columns.

Creating Multiple Connections

When you drop an object on the editing pane from the **File Explorer** window, Stylus Studio displays the connection information in the schema pane of the XQuery Editor. If you then drag and drop another object, Stylus Studio either

- Adds a new connection, if the object was from a different server or port
- Modifies the existing connection, if the object is from the same server and port

A new `collection()` statement is created for each object you drop on the editing pane of the XQuery editor.

How to Create a collection() Statement

◆ **To create a collection() statement:**

1. Open a new XQuery document if one is not already open. The **XQuery Source** tab should be displayed.
2. Ensure that you have established a valid database connection as described in [“Creating a Database Connection”](#) on page 779.
3. In the **File Explorer** window, expand the database and tablespace to display the tables or views you want to access in your XQuery, as shown in this example:

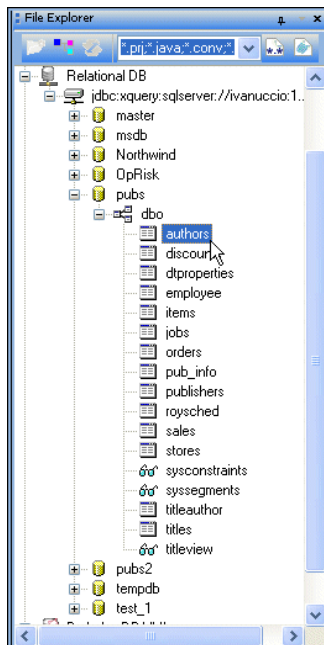


Figure 319. Expanded Database Connection

Optionally, display table and view columns by selecting **Read Structure** from the table or view shortcut menu (right-click).

4. Drag the desired table or view and drop it on the editing pane of the **XQuery Source** tab.

Optionally, drop the table or view on the schema pane of the **XQuery Source** tab. If you do this, you must then drag the desired table or view from the schema pane to the editing pane to create the `collection()` statement.

Stylus Studio creates the `collection()` statement based on the table or view you selected in [Step 4](#). It also displays the table's or view's database in the schema pane of the XQuery Editor (see [Figure 318](#)).

Other Ways to Register a Database Configuration

If you choose, you can explicitly register a database connection by dropping any elements from a database connection displayed in the **File Explorer** window (the connection representation, database, schema, table, or view) on the *schema* pane of the **XQuery Source** tab. You might want to do this when you want to view a table's or view's columns prior to writing your XQuery code, so you can see what data structures are available, as shown here:

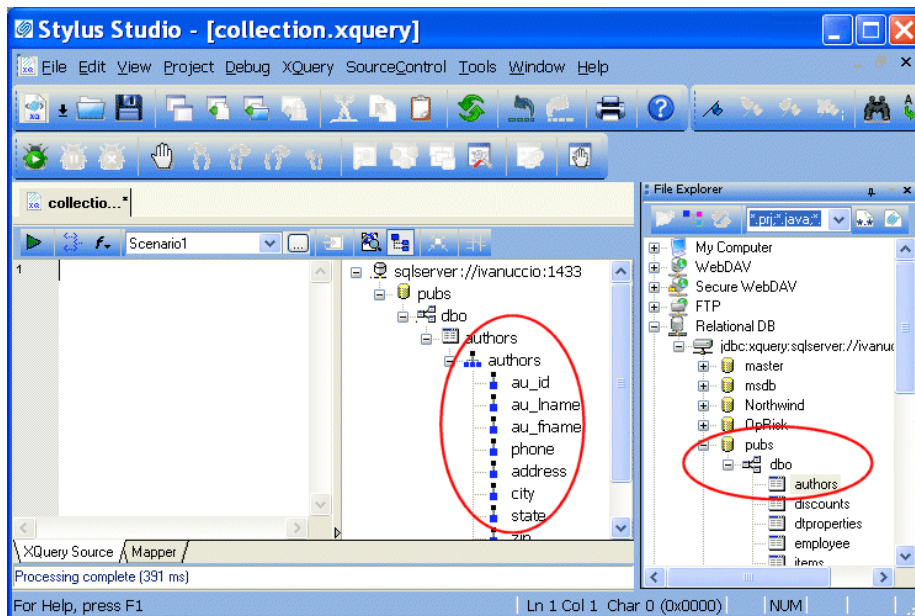


Figure 320. Schema Pane Shows Table and View Columns

This also gives you the ability to close the **File Explorer** window after creating your database connection – providing more room to work, and simplifying the Stylus Studio desktop display.

Tip You can also gain access to column-level information about a database's tables and views directly from the **File Explorer** window by selecting **Read Structure** from the table or view shortcut menu (right-click).

Choosing a Database Object

You can register a database connection by dragging any database object from the **File Explorer** window. The object you select affects which objects you can then query in your XQuery code:

- Server – all of the server's databases and their child tables and views can be queried
- Database – all of the database's tablespaces and their child tables and views can be queried
- Tablespace – any of the tablespace's tables or views can be queried
- Table/View – only the table or view can be queried
- Column – only the column can be queried

Once you register a database connection to the XQuery document, the configuration information remains associated with the XQuery document until you explicitly delete it from the schema pane in the XQuery Editor.

Debugging XQuery Documents

Complex XQuery documents require robust debugging features.

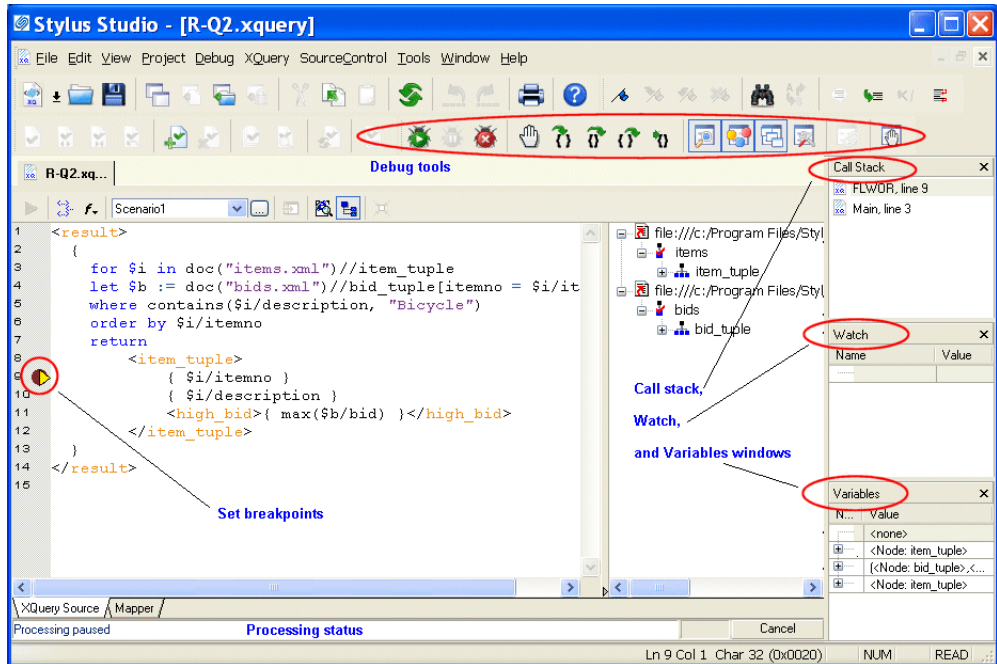


Figure 321. Debugging XQuery in Stylus Studio

With Stylus Studio, you can

- Set breakpoints in your XQuery documents
- Monitor the value of XQuery variables.
- Trace the sequence of XQuery expressions that created output. With a click anywhere in the result, Stylus Studio Visual Backmapping technology displays the XQuery expression responsible for creating that result.

This section covers the following topics:



- [“Using Breakpoints”](#) on page 788
- [“Viewing Processing Information”](#) on page 789
- [“Using Bookmarks”](#) on page 791
- [“Profiling XQuery Documents”](#) on page 792

Using Breakpoints

The Stylus Studio debugger allows you to interrupt XQuery processing to gather information about variables and XQuery expression execution at particular points.

Inserting Breakpoints


◆ **To insert a breakpoint:**

1. In the XQuery document in which you want to set a breakpoint, place your cursor where you want the breakpoint to be.
2. Click **Toggle Breakpoint**  or press F9. Stylus Studio inserts a blank stop sign  to the left of the line with the breakpoint.


Removing Breakpoints





◆ **To remove a breakpoint:**

1. Click in the line that has the breakpoint.
2. Press F9 or click **Toggle Breakpoint**.


Alternative: In the Stylus Studio tool bar, click **Breakpoints**  to display a list of breakpoints in all open files. You can selectively remove one or more, remove them all, or jump to one of them.

Start Debugging

When your XQuery has one or more breakpoints set, start processing by clicking **Start Debugging**  or pressing F5. When Stylus Studio reaches the first breakpoint, it suspends processing and activates the debugging tools. After you examine the information associated with that breakpoint (see [Viewing Processing Information](#) on page 789) you can choose to

- Step into – click  or press F11.
- Step over – click  or press F10.
- Step out – click  or press Shift+F11.
- Run to cursor – click .
- Continue processing – press F5.

- Stop processing – click **Stop Debugging**  in the Stylus Studio tool bar, or click **Cancel** in the lower right corner of the XQuery editor, or press Shift+F5.


Note You can also click **Pause**  to suspend XQuery processing. Stylus Studio flags the line it was processing when you clicked **Pause**.

Viewing Processing Information

Stylus Studio provides several tools for viewing processing information. The tools become active when processing reaches a breakpoint. This section discusses the following topics:

- [“Watching Particular Variables”](#) on page 789
- [“Evaluating XPath Expressions in the Current Processor Context”](#) on page 789
- [“Obtaining Information About Local Variables”](#) on page 790
- [“Displaying a List of Process Suspension Points”](#) on page 790
- [“Displaying XQuery Expressions for Particular Output”](#) on page 790


Watching Particular Variables

Use the **Watch** window to monitor particular variables. To display the **Watch** window, click **Watch**  in the Stylus Studio tool bar. This button is active when Stylus Studio suspends processing because it reached a breakpoint. Stylus Studio displays the **Watch** window only when processing is suspended.


Enter the names of the variables you want to watch. You can enter as many as you like. When Stylus Studio suspends processing, it displays the current values for any variables listed in the **Watch** window. You can expand and collapse complex structures as needed.

During XQuery debugging, you can enter XPath expressions in the **Watch** window fields. Stylus Studio uses the current context to evaluate these expressions, and displays the results with the same kind of interface Stylus Studio uses for `nodeList` and `node` variables.

Evaluating XPath Expressions in the Current Processor Context

When you suspend processing, you can evaluate an XPath expression in the context of the suspended process. You do this in the **Watch** window. Click  in the Stylus Studio tool bar to display the **Watch** window. Click in an empty name field and enter an XPath expression. As soon as you press Enter, Stylus Studio displays the results of the evaluation in the **Value** field of the **Watch** window.

Obtaining Information About Local Variables

Display the **Variables** window to obtain information about local variables. To display the **Variables** window, click **Variables**  in the Stylus Studio tool bar. This button is active when Stylus Studio suspends processing because it reached a breakpoint. Stylus Studio displays the **Variables** window only when processing is suspended.


Information displayed in the **Variables** window includes:

- Information about how the return value (displayed in the **Variables** window as `__Return_Value_3`, for example) is being built
- Local and global XQuery variable values

Also, you can navigate the structure associated with a variable, a parameter, or the current context if it is a node list or a node.

Displaying a List of Process Suspension Points

Display the **Call Stack** window to view a list of the locations at which processing was suspended. For XQuery documents, Stylus Studio displays the XQuery document name and line number.

To display the **Call Stack** window, click **Call Stack**  in the Stylus Studio tool bar. This button is active when Stylus Studio suspends processing because it reached a breakpoint. Stylus Studio displays the **Call Stack** window only when processing is suspended.

When processing is complete, the call stack is empty.

When execution is suspended you can use the **Call Stack** window to jump directly to the XQuery source. Double-click on a stack line to go to that location. A green triangle appears to indicate this location in the source file.

Displaying XQuery Expressions for Particular Output

After you create an XQuery, or during XQuery debugging, Stylus Studio can display the XQuery expression that generated a particular part of a result document. This can be particularly helpful when the result is not quite what you want.


In the **Preview** window, click on the output for which you want to display the XQuery expression. You can do this while either the text view or the browser view is active. Stylus Studio flags the line in the XQuery source that contains the expression that generated the output you selected.

Using Bookmarks

When you are editing or debugging a long file, you might want to repeatedly check certain lines in the file. To quickly focus on a particular line, insert a bookmark for that line. You can insert any number of bookmarks. You can insert bookmarks in any document that you can open in Stylus Studio.

Inserting

◆ **To insert a bookmark:**

1. Click in the line that you want to have a bookmark.
2. Click **Toggle Bookmark**  in the Stylus Studio tool bar. Stylus Studio inserts a turquoise box with rounded corners to the left of the line that has the bookmark.

Removing

◆ **To remove a bookmark:**

1. Click in the line that has the bookmark you want to remove.
2. Click **Toggle Bookmark** in the Stylus Studio tool bar. Stylus Studio removes the turquoise box.

◆ **To remove all bookmarks in a file, click Clear All Bookmarks** .

Moving Focus

◆ **To move from bookmark to bookmark, click Next Bookmark**  **or Previous Bookmark** .

Profiling XQuery Documents



XQuery profiling is available only in Stylus Studio XML Enterprise Suite.

In addition to debugging tools for XQuery, Stylus Studio provides the *XQuery Profiler*, a tool that helps you evaluate the efficiency of your XQuery. By default, the performance metrics gathered by the XQuery Profiler are displayed in a preformatted report, like the one shown here:

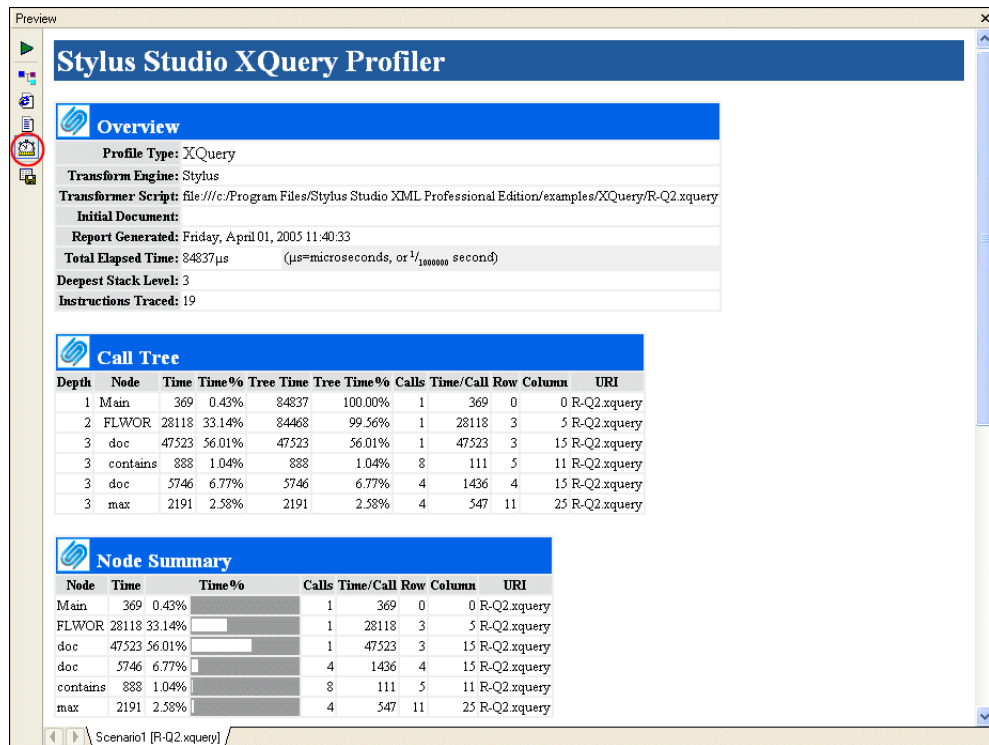


Figure 322. XQuery Profiler Report

The report format is controlled by the default XSLT stylesheet, **profile.xsl**, in the \Stylus Studio\bin directory. You can customize this stylesheet as required. You can save XQuery Profiler reports as HTML.

Note XQuery and XSLT Profiler reports use the same XSLT stylesheet.

In addition to generating the standard XQuery Profiler report, you can save the raw data generated by the Profiler and use this data to create your own reports. See “[Enabling the Profiler](#)” on page 793 for more information about this procedure.



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the XQuery Profiling video](#).

A complete list of the videos demonstrating Stylus Studio’s features is here: http://www.stylusstudio.com/xml_videos.html.

About Performance Metrics

The XQuery Profiler can record three different levels of performance metrics:


- A call tree of execution times
- Execution times by XQuery expression, and
- A detailed log of step-by-step expression execution

Note Displaying the report for a step-by-step log can take significantly longer than evaluating the XQuery itself. Consider using the Profiler with the first two performance metric options. You can also use the **Limit Trace To** fields to further restrict the Profiler’s scope. If you find you need still more detail (while troubleshooting a performance bottleneck, for example), use the step-by-step setting.

Enabling the Profiler

The XQuery Profiler is off by default. You enable the Profiler on the **Profiling Options** tab of the XQuery **Scenario Properties** dialog box.

◆ To enable the XQuery Profiler:

1. Open the **Scenario Properties** dialog box for the XQuery document. (Click **Browse**  at the top of the XQuery editor window.)

2. Click the **Profiling Options** tab.

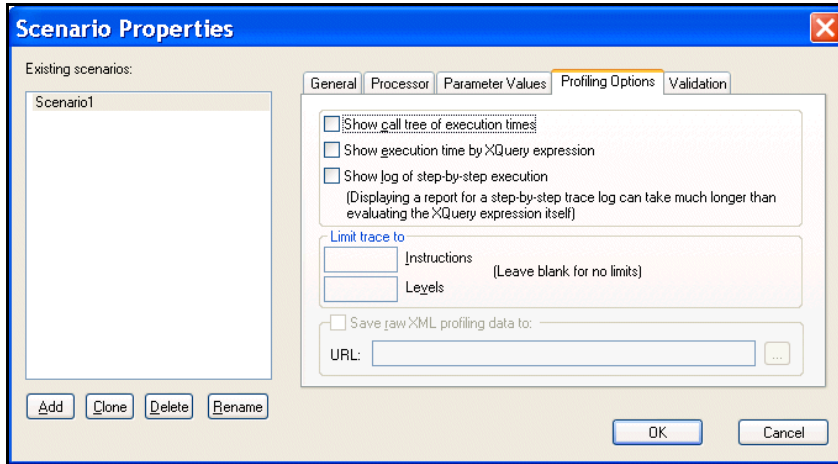


Figure 323. XQuery Profiler Options

3. Select the settings for the performance metrics you want the Profiler to capture.
4. Optionally, save the raw Profiler data to a separate file.

Note



This option is available only after you select one or more performance metrics settings.

5. Click OK.

The next time you preview the XQuery results, the performance metrics you selected are available to you in the XQuery Profiler report (and as raw data if you selected that setting and specified a file).

Displaying the XQuery Profiler Report

◆ **To display the XQuery Profiler report:**

1. Ensure that the Profiler is enabled. (See [“Enabling the Profiler”](#) on page 793 if you need help with this step.)
2. Click the **Preview Result** button ().
3. Click the **Show Profiling Report** button ().

The XQuery Profiler report appears in the **Preview** window.

Using DataDirect XQuery® Execution Plans



DataDirect XQuery execution plan support is available only in Stylus Studio XML Enterprise Suite.

DataDirect XQuery generates an XQuery execution plan so that you can see how DataDirect XQuery will execute your query. For example, if your query accesses a relational data source, the plan will include the SQL statements that DataDirect XQuery will send to the database.

One of the main benefits of using this feature is that you can tune your queries for the best performance possible.

Query Plans in Stylus Studio

In Stylus Studio, a query plan for the DataDirect XQuery® processor becomes available as soon as your XQuery code is well-formed. (Query plans are created only by the DataDirect XQuery® processor.) The query plan changes with your XQuery code – if you add a new data source, for example, the query plan is modified to reflect the new source.

You can view query plans in Stylus Studio to gain insight into how the DataDirect XQuery® processor will execute your XQuery code, including seeing the type of SQL statements that are used to access relational data, when XML streaming is being used, which temporary tables are being created, when variables are being called, and so on. Query plans are displayed on the **Plan** tab of the XQuery Editor. An example of a query plan is shown in [Figure 324](#).

Example of a Query Plan

The example query plan shown in [Figure 324](#) provides information about how DataDirect XQuery translates the following query, which accesses one relational data source, into a SQL Select statement and how XML results are constructed.

```
declare option ddtex:plan-explain 'format=xhtml';
<myHoldings> {
for $holdings in collection("pubs.dbo.holdings")/holdings
where $holdings/userid = "Minollo"
return <holding
quantity="{ $holdings/shares}"}>{ $holdings/stockticker/text()}</holding>
}
</myHoldings>
```

In the following query plan, notice how the Relational Data Source node includes details about the SQL Select statement, as well as information about how the result (\$PT) is constructed.

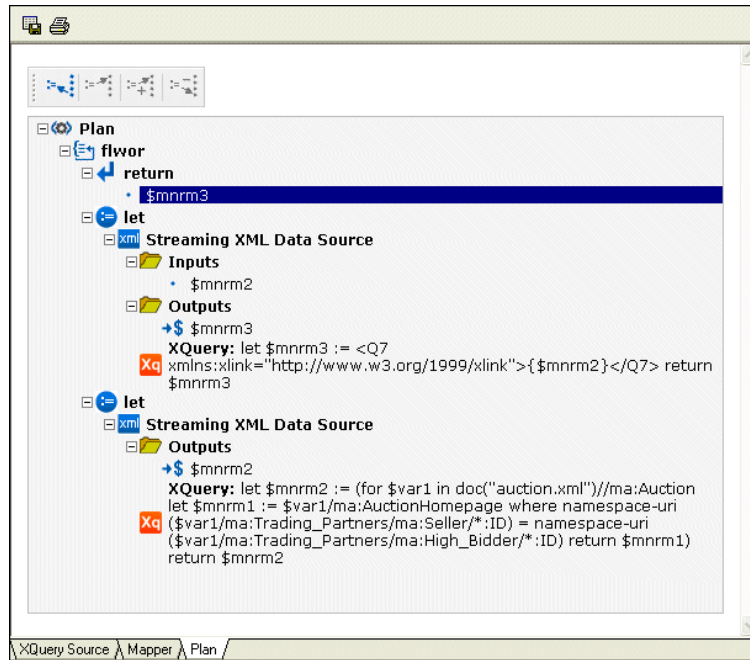


Figure 324. Example of a Query Plan Displayed in the XQuery Editor

Parts of a Query Plan

The graphic representation of a query plan is a tree structure that provides the details of how DataDirect XQuery will execute the query for which the plan was generated. The query plan tree diagram is read-only, though it does provide navigation and formatting features. You can also print and save the query plan as HTML.

In addition to the Plan node, the query plan tree can include Adaptors, Global Variables, and Local Functions nodes. These nodes are described in the following table.

Table 96. Query Plan Nodes

<i>Node</i>	<i>Description</i>
Adaptors	This node contains a list of database resources that will be involved in the execution of the query. These resources can include JDBC connections, temporary tables, and deferred SQL statements used in the context of DataDirect XQuery update functionality.
Global Variables	This node contains a list of global variables that are available to the query plan, such as external variables defined by the query and variables defined as part of the generation of the execution plan.
Local Functions	This node contains a list of user-defined functions used during the query evaluation. Each user-defined function listed in this node has a plan description associated with it. Plan descriptions are described next.
Plan	This node contains the description of the query execution plan. It contains the nodes of the plan, for example, FLWOR nodes and the nodes within the FLWOR nodes such as for, let, and return.

Navigation

You can navigate the tree to check where variables are defined and where they are referenced. For example, you can navigate from one adaptor's definition to its references and vice-versa.





To navigate the tree, you can

- Use the toolbar displayed at the top of the tree
- Right-click an item in the tree and use the context-sensitive menu
- Scroll through the nodes on the tree diagram individually

Query Plan Toolbar

The query plan toolbar has buttons that help you navigate the variables defined in your XQuery code. These buttons are described in the following table.

Table 97. Query Plan Toolbar Buttons

<i>Button</i>	<i>Description</i>
	Go to definition: given a selected variable reference, go to the position in the plan where the variable is defined.
	Go to first reference: given a selected variable definition, go to its first reference in the plan.
	Go to next reference: given a selected variable reference, go to the next reference of the same variable (if any).
	Go to previous reference: given a selected variable reference, go to the previous reference of the same variable (if any).

Formatting

You can change the font size used to display query plan text and symbols by right-clicking a tree node and selecting the font size you wish to use. Changes to font size affect the entire query plan, but they are not saved when you save the query plan as HTML.

Saving a Query Plan as HTML

You can save a query plan as an HTML document; you might wish to do this for review or presentation purposes, for example.

◆ **To save a Query plan as HTML:**

1. Select **XQuery > Save Plan as HTML** from the Stylus Studio menu.
Alternative: Click the **Save Plan** button on the **Plan** tab.
The **Save As** dialog box appears.
2. Choose a path and name for the HTML file.
3. Click **Save**.

Displaying a Query Plan

This section describes the prerequisites and procedure for displaying a query plan in Stylus Studio.

Prerequisites

In order to display a query plan in Stylus Studio, you need to specify the DataDirect XQuery® processor for your XQuery. See “[Selecting an XQuery Processor](#)” on page 802 for more information.

How to display a query plan

◆ **To display a Query plan:**

1. Open the XQuery whose query plan you want to view in the Stylus Studio XQuery Editor.
2. Make sure your XQuery’s processor is set to the DataDirect XQuery® processor. See “[Selecting an XQuery Processor](#)” on page 802 if you need help with this step.
3. Click the **Plan** tab in the XQuery Editor.
Stylus Studio displays the query plan.


Optimizing Your XQuery

One of the main benefits of the query plan is that you can use it when tuning your queries for the best performance possible. After viewing the query plan and executing the XQuery, you might wish to make some changes to the XQuery code to see how they affect the query plan and, consequently, how the XQuery code is processed.

See the *DataDirect XQuery User’s Guide and Reference* for more information on optimizing your XQuery code for data access.

Tip The Stylus Studio Profiler generates an HTML report that contains performance metrics for your XQuery code. You might want to run and view this report before making changes to your XQuery code. See “[Profiling XQuery Documents](#)” on page 792 for more information.

Creating an XQuery Scenario

An *XQuery scenario* is a group of settings you associate with an XQuery. Examples of scenario settings include a main input document, XQuery processors, and whether or not you want to perform validation on the XML that results from your XQuery. Each time you preview an XQuery, Stylus Studio uses settings from the currently active scenario. For example, if the currently active scenario specifies the built-in processor, Stylus Studio executes the XQuery using that processor when you click the **Preview Result** button ()

You can create multiple scenarios for a single XQuery, and choose different settings for each. This flexibility can aid the XQuery development process as it enables you to easily test the XQuery against different input documents and using different processors before making the XQuery available.

This section covers the following topics:

- [“Specifying XML Input”](#) on page 800
- [“Selecting an XQuery Processor”](#) on page 802
- [“Setting Default Options for Processors”](#) on page 804
- [“Setting Values for External Variables”](#) on page 805
- [“Performance Metrics Reporting”](#) on page 806
- [“How to Run a Scenario”](#) on page 809
- [“Working with the XQuery collection\(\) Function”](#) on page 777
- [“How to Create a Scenario”](#) on page 808
- [“How to Run a Scenario”](#) on page 809
- [“How to Clone a Scenario”](#) on page 809

Specifying XML Input

When you create an XQuery scenario, you can optionally specify inputs – XML documents or other sources of XML that set the context for the XPath expressions in your XQuery code. This XML source is referred to as the main input.

The *main input* is a URL for a specific XML document. Specifying a main input is an alternative to using the XQuery `document ()` function in your XQuery code. When you specify a main input document, expressions like `\books\book` in your XQuery code are evaluated in the context of that document.

You specify XML input on the **General** tab of the **Scenario Properties** dialog box.

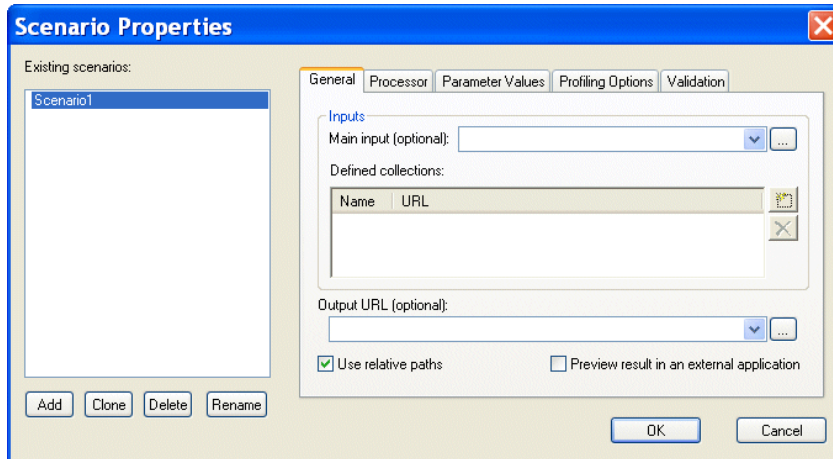


Figure 325. XQuery Scenario General Properties

Note If you build your XQuery using the XQuery Mapper, Stylus Studio uses the first source document you select as the main input XML document, though you can override this default at any time. See “[Source Documents](#)” on page 752 to learn more about the process of selecting and working with XQuery source documents in XQuery mapper.

Selecting an XQuery Processor

You use the **Processors** tab of the **Scenario Properties** dialog box to specify the processor you want to use to process your XQuery code.

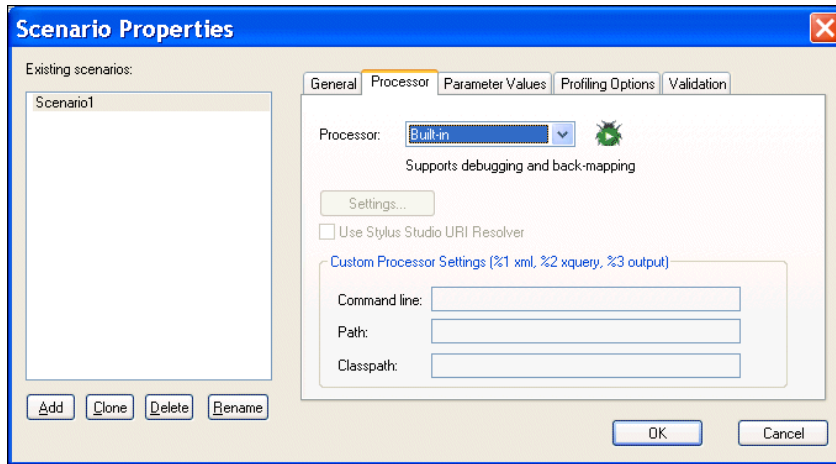


Figure 326. XQuery Scenario Processor Properties

You can use

- The DataDirect XQuery[®] processor
- One of several processors that support XQuery debugging and backmapping, including Stylus Studio's built-in XQuery processor and Saxon
- An external instance of the Stylus Studio processor
- The Raining Data[™] TigerLogic[™] XDMS XQuery processor, which runs on the TigerLogic XDMS server
- Any custom processor you specify

Tip You can define default settings for XQuery processors, and you can also choose to use one of them as the default XQuery processor. See [“Setting Default Options for Processors”](#) on page 804.

Using the Saxon Processor

Stylus Studio lets you execute XQuery documents using either the Saxon-B (basic) or Saxon-SA (schema-aware) processor. You specify which processor you want to use with the **Execution mode** property in the **Saxon XQuery Settings** dialog box. Settings that have

command line equivalents in Saxon show the command in parentheses following the property name. Some settings are available only if you are using Saxon-SA.



Support for Saxon-SA is available only in Stylus Studio XML Enterprise Suite.

Stylus Studio's Sense:X syntax coloring and auto-completion provides full support for Saxon syntax, so long as the Saxon XQuery processor is either associated with the current XQuery scenario or has been set as the default XQuery processor.

If you want to use the Saxon processor:

1. On the **Processors** tab, click **Saxon**.
The **Settings** button becomes active.
2. Click the **Settings** button.
The **Saxon XQuery Settings** dialog box appears.

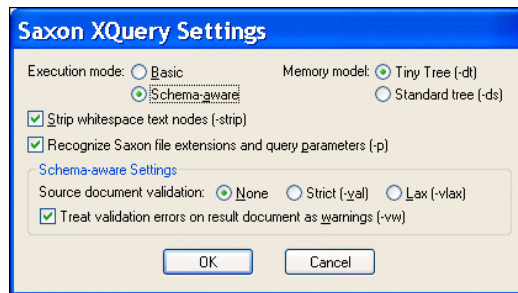


Figure 327. Saxon XQuery Settings Dialog Box

3. Complete the settings as desired. Press F1 to access the Stylus Studio online help, or refer to the [Saxon documentation](#) for more information.
4. Click **OK**.

Using the TigerLogic XDMS Processor

Stylus Studio's Sense:X syntax coloring and auto-completion provides full support for TigerLogic XDMS syntax, so long as the TigerLogic XDMS XQuery processor is either associated with the current XQuery scenario or has been set as the default XQuery processor.

If you want to use the TigerLogic XDMS processor:

1. Click **TigerLogic XDMS**.

The **Settings** button becomes active.

2. Click the **Settings** button.

The **TigerLogic XDMS Server Settings** dialog box appears.

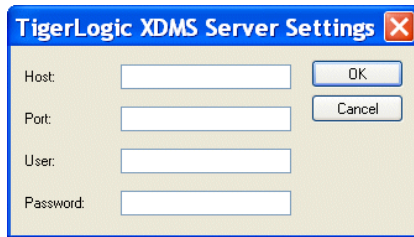


Figure 328. TigerLogic XDMS Server Settings Dialog Box

3. Enter the host, port, username, and password information for the server on which the TigerLogic XDMS is running.
4. Click **OK**.

Setting Default Options for Processors

You can set default values for all XQuery processors on the **Processor Settings** page of the **Options** dialog box. In addition, you can select a processor other than the built-in processor as the default XQuery processor.

You can always override the default processor and default processor settings at the scenario level.

◆ To set defaults for XQuery processors:

1. From the Stylus Studio menu, select **Tools > Options**.
Stylus Studio displays the **Options** dialog box.

2. Select **Module Settings > XQuery > Processor Settings**.

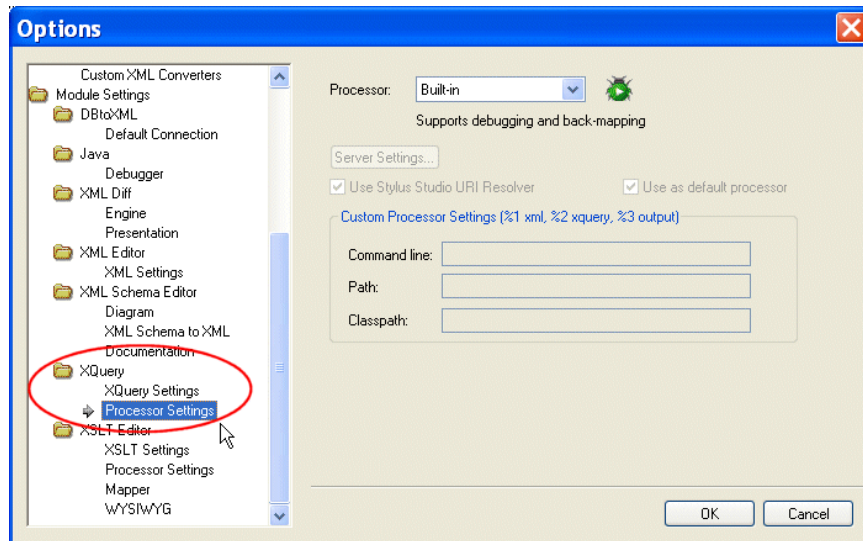


Figure 329. Options for XQuery Processors

3. Select the processor for which you want to specify default settings from the **Processor** drop-down list.
4. If required, complete processor-specific settings. (Click the **Settings** button.)
5. If you want to use this processor as the default processor for all XQuery scenarios, click the **Use as default processor** check box.
6. Click **OK**.

Setting Values for External Variables

The **Parameter Values** tab of the **Scenario Properties** dialog box displays any external variables you have defined in the XQuery source. You can specify the parameter value you want to use for any external variables when you run the scenario. For example, imagine your XQuery code contains the following:

```
declare variable $part_num external
```

This variable is displayed on the **Parameters** tab as follows:

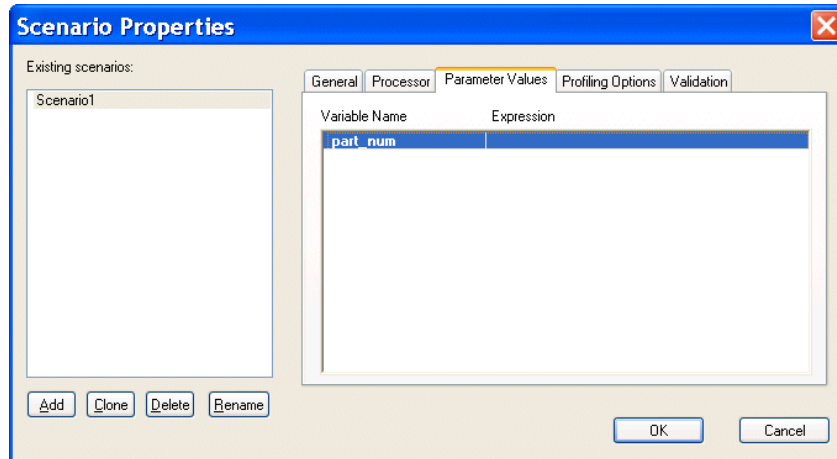


Figure 330. XQuery Scenario Parameters

When you run the scenario, you can specify the parameter value you want to use by double-clicking the **Expression** field and typing a value. Valid values are XPath expressions and must be entered using single or double quotes.

Performance Metrics Reporting


See “[Enabling the Profiler](#)” on page 793 to learn more about the different ways in which Stylus Studio can provide you with XQuery performance metrics.

Validating XQuery Results

You can optionally validate the XML document that results from XQuery processing. You can validate using the

- Stylus Studio built-in processor. If you use the Stylus Studio built-in processor, you can optionally specify one or more XML Schemas against which you want the result document to be validated.
- Any of the customizable processors supported by Stylus Studio, such as the .NET XML Parser and XSV.

◆ **To validate XQuery scenario result documents:**

1. Open the XQuery whose results you want to validate.
2. In the XQuery Editor, in the scenario name field, click the down arrow and click the name of the scenario for which you want to perform validation.
3. Click **Browse**  to open the **Scenario Properties** dialog box.
4. Click the **Validation** tab.

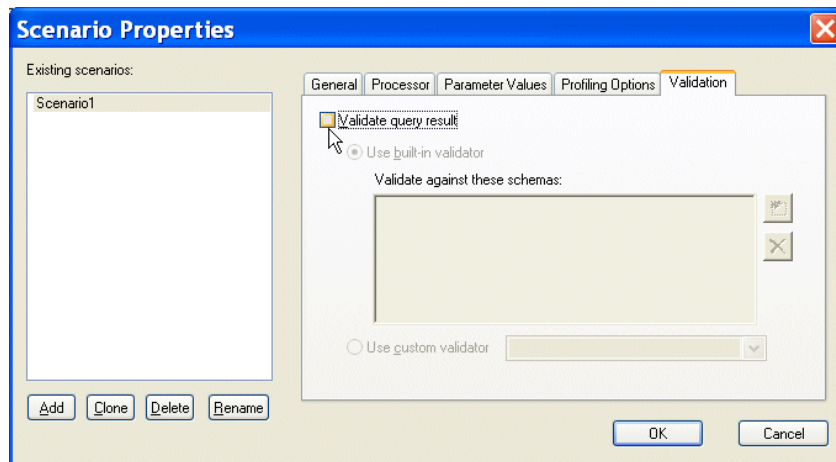





Figure 331. Validation Tab for XQuery Scenarios

5. Click **Validate query result**.
6. If you are using Stylus Studio's built-in validation engine, optionally, specify the XML Schemas against which you want to validate the XML result document. Otherwise, go to [Step 7](#)
 - a. Click the Open file button ().
The **Open** dialog box appears.
 - b. Select the XML Schema you want to use for validation.
 - c. Click the **Open** button to add the XML Schema to the **Validation** tab.
 - d. Optionally, add other XML Schemas.
 - e. Go to [Step 8](#).

7. Click the **Use custom validator** button, and select the validation engine you want to use from the drop-down list box.
8. Click **OK**.

How to Create a Scenario

◆ To create a scenario:

1. In the XQuery editor tool bar, click .
Alternative: Select **Create Scenario** from the scenario drop-down list at the top of the editor window.
Stylus Studio displays the **Scenario Properties** dialog box.
2. In the **Scenario name:** field, type the name of the new scenario.
3. In the **Main input:** field, type the name of the XML file to which you want to apply the XQuery, or click **Browse**  to navigate to an XML file and select it.

Note

If the first document you added to the XQuery is an XML document, Stylus Studio uses that document as the XML source for the scenario and displays it in this field.

4. If you are using DataDirect XQuery[®], specify one or more defined collections as input. See [“Specifying XML Input”](#) on page 800 if you need help with this step.
5. In the **Output URL** field, optionally type or select the name of the result document you want the XQuery document to generate. If you specify the name of a file that does not exist, Stylus Studio creates it when you preview the XQuery.
6. If you want Stylus Studio to store paths relative to XQuery document path, ensure that the **Use relative paths** option is checked.
7. If you check **Preview result in an external application**, Stylus Studio displays the result Internet Explorer. In addition, Stylus Studio always displays XQuery results in the **Preview** window.
8. If you want to specify values for XQuery parameters, click the **Parameter Values** tab. Click the **Variable Name** field for the parameter – Stylus Studio places the text cursor in the **Expression** field, allowing you to type a value for the parameter.
9. If you want Stylus Studio to capture performance metrics, enable the XQuery Profiler on the **Profiling Options** tab. See [Profiling XQuery Documents](#) on page 792.

10. To define another scenario, click **Add** and enter the information for that scenario. You can also copy scenarios. See [How to Clone a Scenario](#) on page 809.

11. Click **OK**.


If you start to create a scenario and then change your mind, click **Delete** and then **OK**.

How to Run a Scenario

◆ To run a scenario:

1. Select a scenario from the scenario drop-down list at the top of the editor window.

Alternative:


- a. In the XQuery editor tool bar, click .
Stylus Studio displays the **Scenario Properties** dialog box.
- b. On the **General** tab, select the scenario you want to run from the **Existing Scenarios** list.
- c. Click **OK**.

2. Click the **Preview Result** button (.

How to Clone a Scenario

When you clone a scenario, Stylus Studio creates a copy of the scenario except for the scenario name. This allows you to make changes to one scenario and then run both to compare the results.

◆ To clone a scenario:

1. Display the XQuery in the scenario you want to clone.
2. In the XQuery editor tool bar, click  to display the **Scenario Properties** dialog box.
3. In the **Scenario Properties** dialog box, in the **Existing preview scenarios** field, click the name of the scenario you want to clone.
4. Click **Clone**.
5. In the **Scenario name** field, type the name of the new scenario.

6. Change any other scenario properties you want to change. See [How to Create a Scenario](#) on page 808.
7. Click **OK**.

If you change your mind and do not want to create the clone, click **Delete** and then **OK**.

Generating XQuery Documentation

Stylus Studio allows you to generate HTML documentation for your XQuery using xqDoc, from <http://www.xqdoc.org>. This section describes how to generate XQuery documentation, and how to annotate your XQuery code for reporting purposes.

Documentation Defaults

By default, xqDoc generates Module URI, Function Summary, and Function Detail sections for each XQuery, as shown in the following illustration.

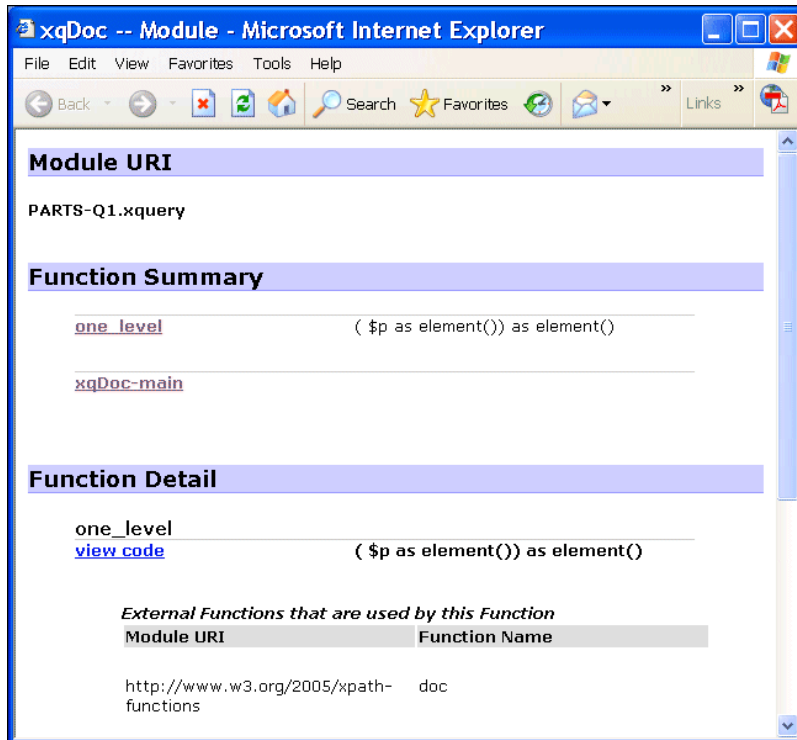


Figure 332. Default xqDoc Report

If you want, you can add your own comments using the syntax described in the following section, “[Syntax and Usage](#)” on page 811.

Syntax and Usage

Comments for xqDoc must start with (:~ and end with :). Comments can span multiple lines. New lines do not need to be introduced with a special character. You can use `
` to force a line break, which you might want to do to aid readability. Comments that start with (: (standard XQuery syntax) are ignored by xqDoc.

The first comment in an XQuery document is interpreted by xqDoc as the Module Description. Within that comment, xqDoc recognizes certain keywords preceded by the

at (@) sign. Examples include @author and @version. See the xqDoc documentation at <http://www.xqdoc.org> for more information. Here is a report for the same document shown in Figure 332 with a user-defined module and function descriptions.

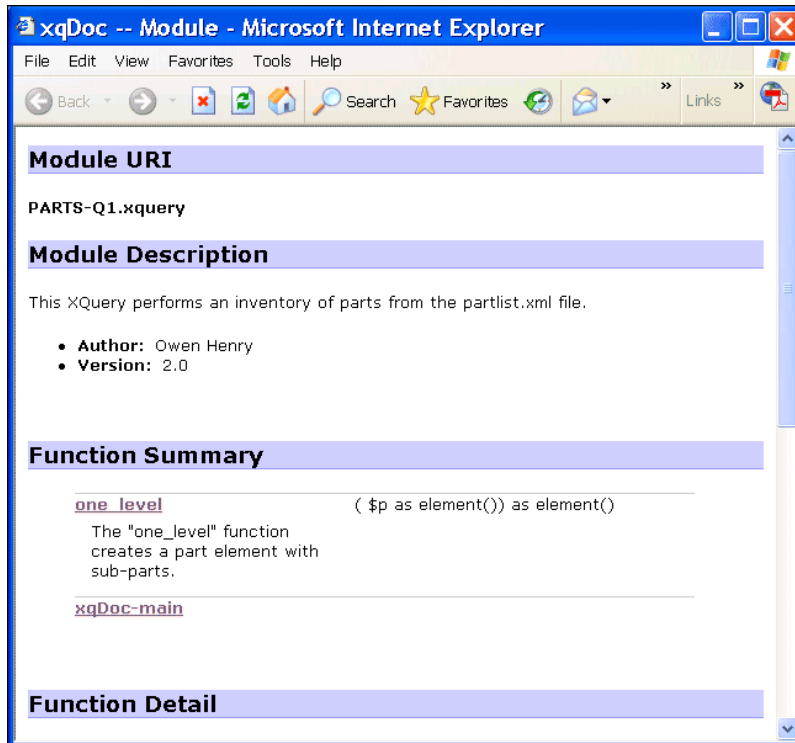


Figure 333. xqDoc Report with Additional Module and Function Descriptions

All other comments must precede function declarations. xqDoc uses the text you enter to provide a description for each function listed in the Function Summary. The same

description is used in the Function Detail. Here is an illustration of the XQuery document in the XQuery Editor; the xqDoc comments are highlighted:

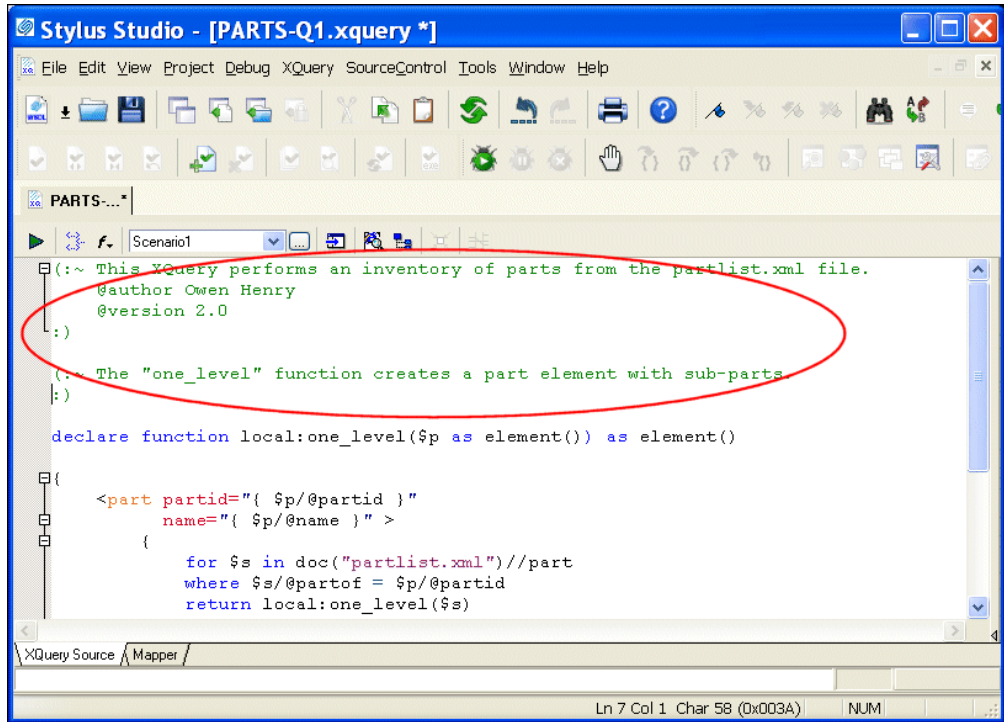


Figure 334. xqDoc Comments as Seen in XQuery Source

Save the XQuery Document

When you annotate an XQuery document using xqDoc comments, make sure to save the XQuery document before generating documentation. Unsaved work is not detected by the report generating mechanism.

ActiveX Controls

xqDoc reports use ActiveX controls for navigation and code sample generation. Make sure to enable ActiveX controls for the browser displaying the xqDoc report.

Viewing Code Samples

You can view code samples from an xqDoc report by clicking the **view code** hyperlink, as shown in the following illustration.

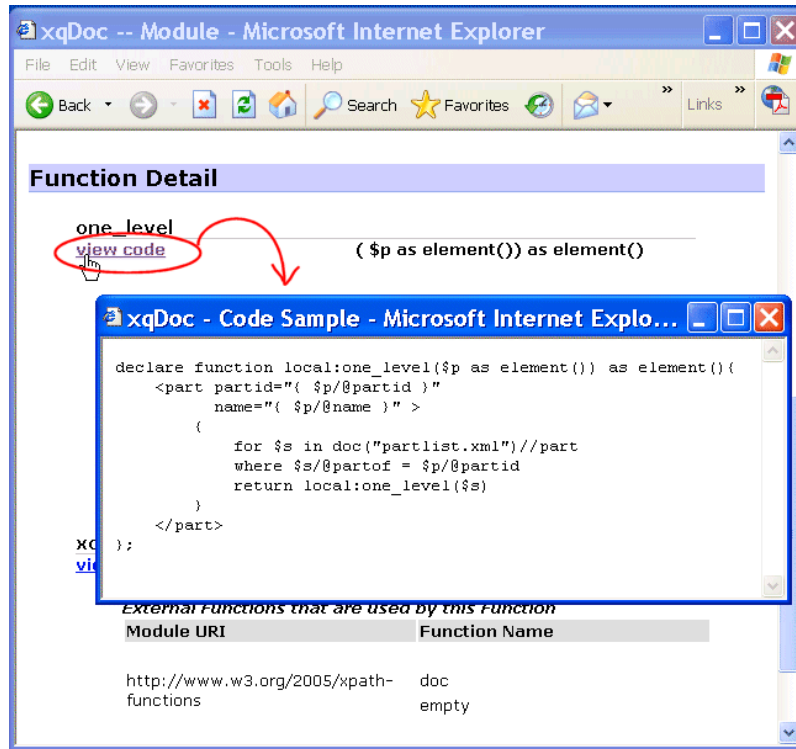


Figure 335. Viewing an XQuery Code Sample

The XQuery code sample appears in a separate Web browser.

How to Generate XQuery Documentation

◆ **To generate XQuery documentation:**

1. Optionally, annotate your XQuery code as described in “Syntax and Usage” on page 811.
2. Save the XQuery document.

3. Click **XQuery > Generate xqDoc**.

Stylus Studio displays the **Browse for Folder** dialog box, which allows you to choose where you want to save the XQuery documentation.

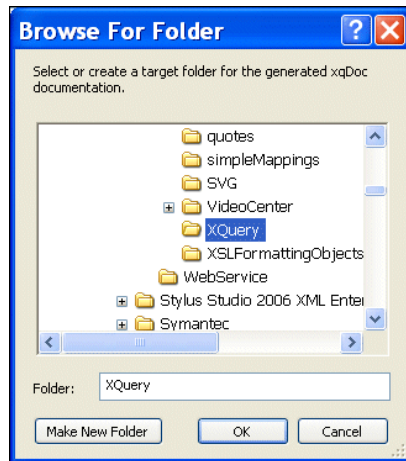


Figure 336. Browse for Folder Dialog Box

By default, Stylus Studio selects the directory to which the XQuery document is saved. After that, Stylus Studio uses the last location to which you saved the XQuery documentation.

4. Optionally, change the location to which you want to save the XQuery documentation.
5. Click **OK**.

Stylus Studio displays processing information in the **Output** window. A new Internet browser is launched; the XQuery documentation is displayed in this browser.

6. If you have not already done so, enable ActiveX controls for the browser window displaying the xqDoc report.

Using XQuery to Invoke a Web Service



Web Service support is available only in Stylus Studio XML Enterprise Suite .

This section describes how to use the XQuery code that Stylus Studio generates from a Web service call. To learn more about the process of generating XQuery from a Web service call, see [“Generating XQuery from a Web Service Call”](#) on page 844.

This section covers the following topics:

- [“Requirements”](#) on page 816
- [“Using the Saxon Processor”](#) on page 816
- [“Invoking a SOAP Request in an XQuery”](#) on page 817
- [“Invoking Multiple SOAP Requests”](#) on page 818

Requirements

In order to use the XQuery generated by Stylus Studio to invoke a Web service, you need to

- Use either the built-in, Saxon, or DataDirect XQuery[®] processor
- Ensure axis-all.jar and xercesimpl.jar are in your classpath. If you are using Java 1.4, you must also include xml-apis.jar .

Tip

You can check the version of your Java Virtual Machine on the About Stylus Studio dialog box (**Help > About Stylus Studio**).

Using the Saxon Processor

By default, the XQuery code generated by Stylus Studio is compliant with both the built-in and DataDirect XQuery[®] processors. If you want to use the Saxon XQuery processor, you need to

- Comment the DataDirect namespace and function bindings:

```
declare namespace ws =  
"dtekjava:com.stylusstudio.webservice.SOAPCall";  
declare function ws:call($location as element(), $payload as  
element()) as document-node() external;
```

- Uncomment the Saxon function binding:

```
declare namespace ws = "java:com.stylusstudio.webservice.SOAPCall";
```

Function bindings for both processor types are preceded with a string that identifies them, similar to this one for Saxon:

```
(: Saxon function binding :)
```

Do not uncomment these strings.

Invoking a SOAP Request in an XQuery

◆ To invoke a SOAP request in an XQuery:

1. Create a new Web service call and configure the SOAP request as required. See [“How to Compose a Web Service Call”](#) on page 828 if you need help with this step.
2. Generate XQuery code from the Web service call. See [“How to Generate XQuery from a Web Service Call”](#) on page 845 if you need help with this step.
As a result of this step, XQuery code is copied to your system’s clipboard.
3. Open a new XQuery (**File > New > XQuery File**). Make sure you are using either the built-in, Saxon, or DataDirect XQuery[®] processor. See [“Selecting an XQuery Processor”](#) on page 802 if you need help with this step.
4. Paste the clipboard contents into the XQuery document.
5. If necessary, modify your system’s Classpath as described in [“Requirements”](#) on page 816.
6. Preview the XQuery by clicking the **Preview Result** button at the top of the XQuery Editor.
The results of the SOAP request contained in the XQuery appear in the **Preview** window.
7. If you are satisfied with the results, save the XQuery.

Invoking Multiple SOAP Requests

You can invoke multiple SOAP requests in the same XQuery. These SOAP requests can be from the same Web service, or from different Web services if they use the same parameters.

Rules

When invoking multiple SOAP requests in the same XQuery, bear in mind the following rules:

- The XQuery must contain only one instance of the header that declares the namespace and function bindings. In other words, copy only the Java extension function, `ws:call()`, to the XQuery.
- Separate each `ws:call()` with a comma, to create a sequence.

How to Invoke Multiple SOAP Requests in the Same XQuery

◆ **To invoke multiple SOAP requests in the same XQuery:**

1. Make sure you understand the rules for including multiple SOAP requests in the same XQuery code as described in the previous section, “[Rules](#)” on page 818.
2. Create the first Web service and generate XQuery code for the SOAP request as described in “[Generating XQuery from a Web Service Call](#)” on page 844.
The XQuery code generated by Stylus Studio is copied to your system’s clipboard.
3. Create a new XQuery, and paste the contents of your system clipboard into it.
4. Type a comma at the end of the `ws:call()` function.
5. Repeat [Step 2](#) for the next Web service SOAP request you want to invoke from your XQuery.
6. Paste the new XQuery code into the XQuery you created in [Step 3](#). Before pasting, place the text cursor after the comma you typed in [Step 4](#).
7. From the XQuery code you just pasted, delete everything except the `ws:call()` function.
8. If you want to add another SOAP request to your XQuery, type a comma after the `ws:call()` function you just pasted, and return to [Step 2](#).

Generating Java Code for XQuery



Java code generation is available only in Stylus Studio XML Enterprise Suite

Stylus Studio includes a Java Code Generation wizard that creates Java code based on the scenarios defined for an XQuery document. This section describes scenario settings that affect the generated code, as well as procedures for generating, compiling, and running generated code.



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the Java Code Generation video](#).

A complete list of the videos demonstrating Stylus Studio's features is here: http://www.stylusstudio.com/xml_videos.html.

This section covers the following topics:

- “[Scenario Settings](#)” on page 820
- “[Java Code Generation Settings](#)” on page 822
- “[How to Generate Java Code for XQuery](#)” on page 822
- “[Compiling Generated Code](#)” on page 824
- “[Deploying Generated Code](#)” on page 826

Tip You can also generate Java code for XSLT. See “[Generating Java Code for XSLT](#)” on page 401.

Scenario Settings

Stylus Studio generates Java code based on the scenarios you have defined for an XQuery document. The following tables summarizes the scenario settings that have an effect on Java code generation.

Table 98. Scenario Settings that Affect Java Code Generation

<i>Tab</i>	<i>Comment</i>
General	The Java Code Generation wizard uses only the Main input and Output URL fields, if specified.
Processor	You must use the Saxon or DataDirect XQuery [®] processor. The Stylus Studio URI Resolver is also used to resolve non-standard URIs if you select the check box on the Processor page.
Parameter Values	Any parameter values you specify are processed by the Java Code Generation wizard.
Profiling Options	Ignored.
Validation	Validation is optional. If you choose to validate XQuery output, the Java Code Generation wizard always uses the built-in Java processor, regardless of the validator you specify on the Validation tab. If you want to specify external schemas for validation purposes, click Use built-in validator . Note that the built-in Java processor is used for validation even in this case.

Choosing Scenarios

You can generate Java code for one or more of the scenarios defined for a single XQuery document. By default, the Java Code Generation wizard selects only the current scenario for generation, as shown in [Figure 337](#).

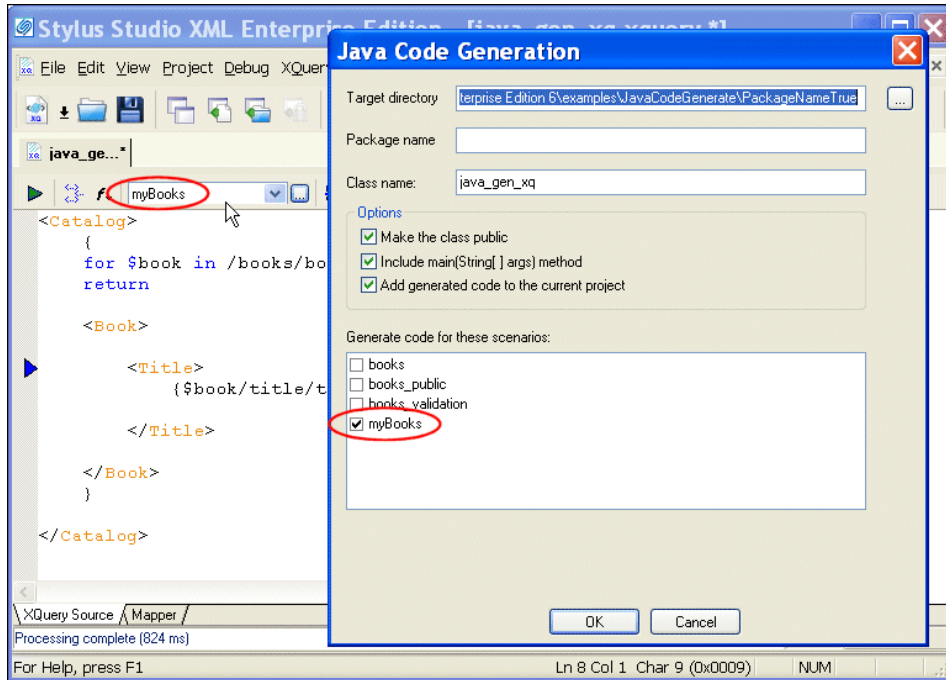


Figure 337. The Current Scenario Is Selected for Code Generation

The generated code includes a `setScenario` method for every scenario you select for code generation, as shown in the following example.

```
// Uncomment the setScenario call for the scenario you want the code to run.
// All other scenarios must be commented out.
// app.setScenario_books();
// app.setScenario_books_public();
// app.setScenario_books_validation();
// app.setScenario_myBooks();
```

Only one `setScenario` method can be called at a time. Uncomment the `setScenario` method for the scenario you want the code to process, and make sure that all other `setScenario` methods are commented.

Java Code Generation Settings

When you generate Java code for an XQuery document, you need to specify

- The target directory in which you want the Java code created. `c:\temp\myJavaCode`, for example. If the directory you name does not exist, Stylus Studio creates it when you run the Java Code Generation wizard. The default is a `\sources` directory, which is created where you installed Stylus Studio when you generate the code, `c:\Program Files\Stylus Studio\sources`, for example.
- Optionally, a package name. If you specify a package name, this name is used for a subfolder created in the target directory you specify. If you specify `myPackage` as the package name, for example, the generated code is written to `c:\temp\myJavaCode\myPackage`. (Though optional, it is considered good practice to create a package name.)
- The class name. Stylus Studio also uses the class name for the `.java` file created by the Java Code Generation wizard. For example, if you provide the name `myClass`, Stylus Studio creates `c:\temp\myJavaCode\myPackage\myClass.java`.

In addition, you can specify whether or not you want to

- Create the class as a public class
- Include the `main(String[] args)` method
- Add the generated code to the current project

All of these options are selected by default.

Tip If you choose to add the generated code to the project, it creates a folder using the package name you specify and places the `.java` file in that folder. If you do not specify a package name, the `.java` file is added directly below the project root in the **Project** window.

How to Generate Java Code for XQuery

◆ **To generate Java code for XQuery:**

1. Define at least one scenario for the XQuery document for which you want to generate Java code. The scenario must use the Saxon or DataDirect XQuery[®] processor. See “[Scenario Settings](#)” on page 820 for more information.

2. Select **XQuery > Generate Java Code** from the Stylus Studio menu.
The **Generate Java Code** dialog box appears.

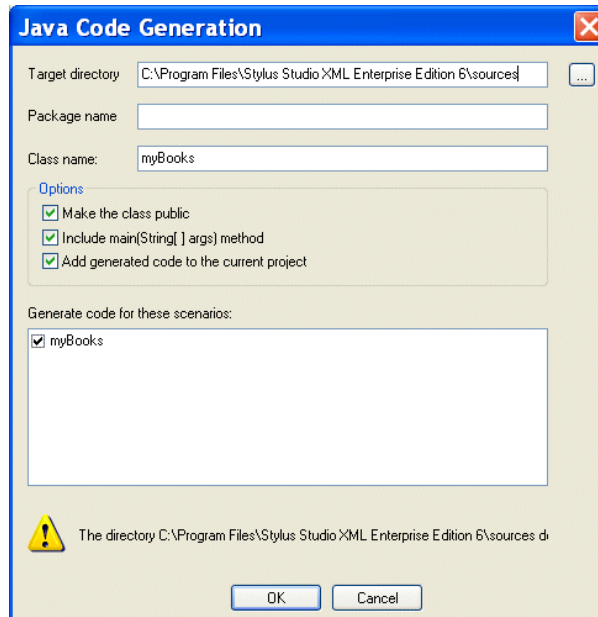


Figure 338. Java Code Generation Dialog Box

3. Specify the settings you want for the target directory, package and class names, and so on. See [“Java Code Generation Settings”](#) on page 822 if you need help with this step.
4. Select the scenarios for which you want to generate Java code.
5. Click **OK**.

Stylus Studio generates Java code for the XQuery. When the code generation is complete, the resulting file (*classname.java*) is opened in the Stylus Studio Java Editor.

Compiling Generated Code

In order to compile the Java code generated for XQuery, you need to make sure that the following JAR files are in the Stylus Studio classpath:

- CustomFileSystem.jar
- Saxon8.jar


These files are in in the \bin directory where you installed Stylus Studio.

How to Modify the Stylus Studio Classpath

◆ **To modify the Stylus Studio classpath:**

1. Select **Tool > Options** from the Stylus Studio menu.
2. Click **Java Virtual Machine** under **General**.
3. Click the browse button alongside the **Classpath** field.

The **Add Directory or JAR File to Classpath** dialog box appears.

4. Click the browse folders () button.

A new entry field appears in the **Locations** list box. Two buttons appear to the right of the entry field.

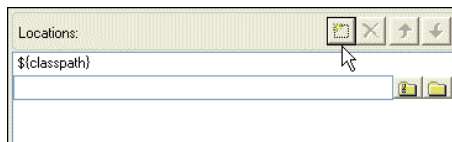


Figure 339. Entry Field for JAR File Classpath


- Click the browse jar files button ().
Stylus Studio displays the **Browse for JAR Files** dialog box.




Figure 340. Browse for JAR Files Dialog Box


- Navigate to the Stylus Studio \bin directory, select the CustomFileSystem.jar file, then click **Open**.
- Repeat [Step 6](#), this time selecting the Saxon8.jar file.
- Click **OK** to close the **Add Directory or JAR File to Classpath** dialog box.

How to Compile and Run Java Code in Stylus Studio

◆ To compile Java code in Stylus Studio:

- Make sure the Java Editor is the active window.
- Click the **Compile** button ().
Alternatives: Press Ctrl + F7, or select **Java > Compile** from the Stylus Studio menu.
Stylus Studio compiles the Java code. Results are displayed in the **Output** window.

◆ To run Java code in Stylus Studio:

- Make sure the Java Editor is the active window.
- Click the **Run** button ().
Alternatives: Press Ctrl + F5, or select **Java > Run** from the Stylus Studio menu.
If the code has not been compiled, Stylus Studio displays a prompt asking if you want to compile the code now. Otherwise, Stylus Studio runs the Java code. Results are displayed in the **Output** window.

Deploying Generated Code

If your XQuery uses built-in DataDirect XML Converters™ – to convert CSV or EDI to XML, for example – you need to purchase licenses for the DataDirect XML Converters you wish to use if you wish to deploy your code in any environment on a machine (such as a test or application server) that does not have a license for the DataDirect XML Converters. Licenses for DataDirect XML Converters are purchased separately from Stylus Studio 2007 XML Enterprise Suite.

Similarly, if you use the DataDirect XQuery® processor, you must acquire additional licences if you wish to deploy your XQuery code.

Write Stylus Studio at stylusstudio@stylusstudio.com, or call 781.280.4488 for more information.

Chapter 11 **Composing Web Service Calls**



The Web services features described in this chapter are available only in Stylus Studio XML Enterprise Suite.

Using Stylus Studio's Web service call composer, you can design, compose, and test a Web service call without writing any code. Once Stylus Studio composes the Simple Object Access Protocol (SOAP) request and you have successfully tested it, you can use the SOAP response returned by the Web service as an XML source wherever you use XML documents in Stylus Studio.



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the Introduction to the Web Service Call Composer video](#).

A complete list of the videos demonstrating Stylus Studio's features is here: http://www.stylusstudio.com/xml_videos.html.

This chapter covers the following topics:

- “Overview” on page 828
- “Obtaining WSDL URLs” on page 831
- “Modifying a SOAP Request” on page 835
- “Testing a Web Service” on page 837
- “Saving a Web Service Call” on page 839
- “Generating a Java Web Service Client” on page 842
- “Generating XQuery from a Web Service Call” on page 844
- “Creating a Web Service Call Scenario” on page 846

Overview

The process of composing a Web service call in Stylus Studio involves the following steps:

1. Specify the Web Services Description Language (WSDL) URL associated with the Web service you want to use. See [“Obtaining WSDL URLs”](#) on page 831.
2. Compose the Simple Object Access Protocol (SOAP) request.
 - a. Select the operation described by the WSDL for which you want Stylus Studio to compose a SOAP request.
 - b. Provide values for the SOAP request parameters.
See [“Modifying a SOAP Request”](#) on page 835.
3. Test the Web service. You can test a Web service call as you composed it, or you can create a scenario to test the Web service call using parameters of your choosing. See [“Testing a Web Service”](#) on page 837.

Once you are satisfied with the Web service call, you can optionally

- Save the Web service call for later use. See [“Saving a Web Service Call”](#) on page 839.
- Generate a Java Web service client. See [“Generating a Java Web Service Client”](#) on page 842.
- Create a Web service scenario. See [“Creating a Web Service Call Scenario”](#) on page 846.

How to Compose a Web Service Call

◆ **To compose a Web Service call:**

1. From the Stylus Studio menu bar, select **File > New > Web Service Call**.

Stylus Studio opens a new document in the Web Service Call Composer.

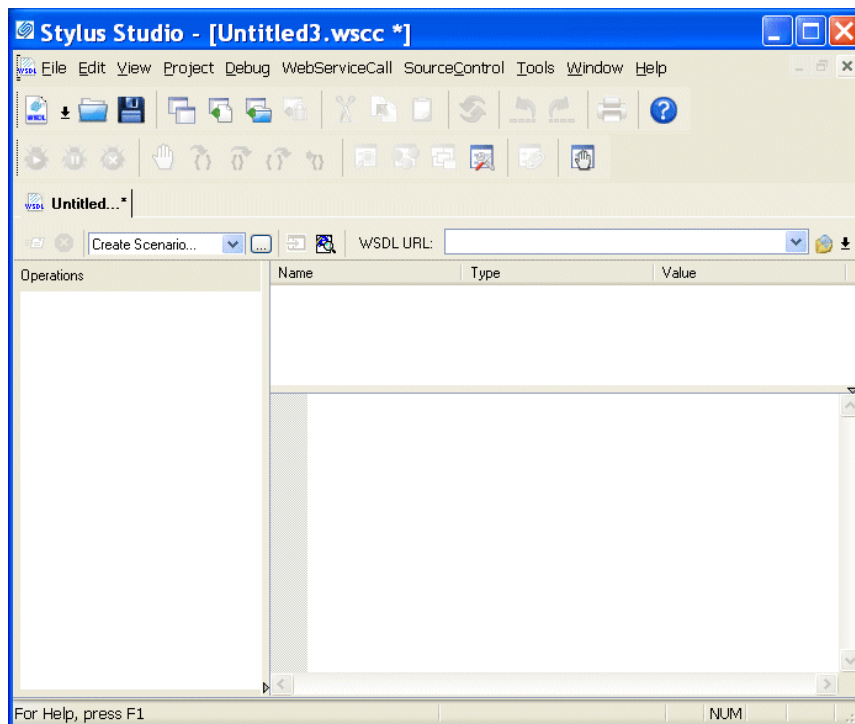


Figure 341. Web Service Call Composer

2. Type a WSDL address in the **WSDL URL** field, or use the **UDDI** button to browse UDDI registries for published Web services. See “[Obtaining WSDL URLs](#)” on page 831 for help with this step. (Any WSDL URLs that you have used previously are displayed in the **WSDL URL** drop-down list.)

Web service operations for the WSDL you select are displayed in the **Operations** field.

3. Select the Web service operation for which you want to create a SOAP request from the **Operations** field.


Parameters for the operation you select are displayed in the **Name** field; the datatype for each parameter is displayed in the **Type** field. The SOAP request is displayed beneath the fields you use to define the operation’s parameters.

4. Set values for the parameters:
 - a. Click the parameter name.

- b. Type a value in the **Value** field.
Stylus Studio updates the SOAP request to reflect the parameter values you enter.

Alternative: You can manually edit the XML in the SOAP request. If you do, the **Value** field is updated automatically.

See “[Modifying a SOAP Request](#)” on page 835 for help with this step.

5. When you have provided values for all of the parameters, click the **Send Request** button () to test the Web service.

If it is not already open, Stylus Studio opens the **Preview** window and displays the SOAP response returned by the Web service, as shown in [Figure 342](#):

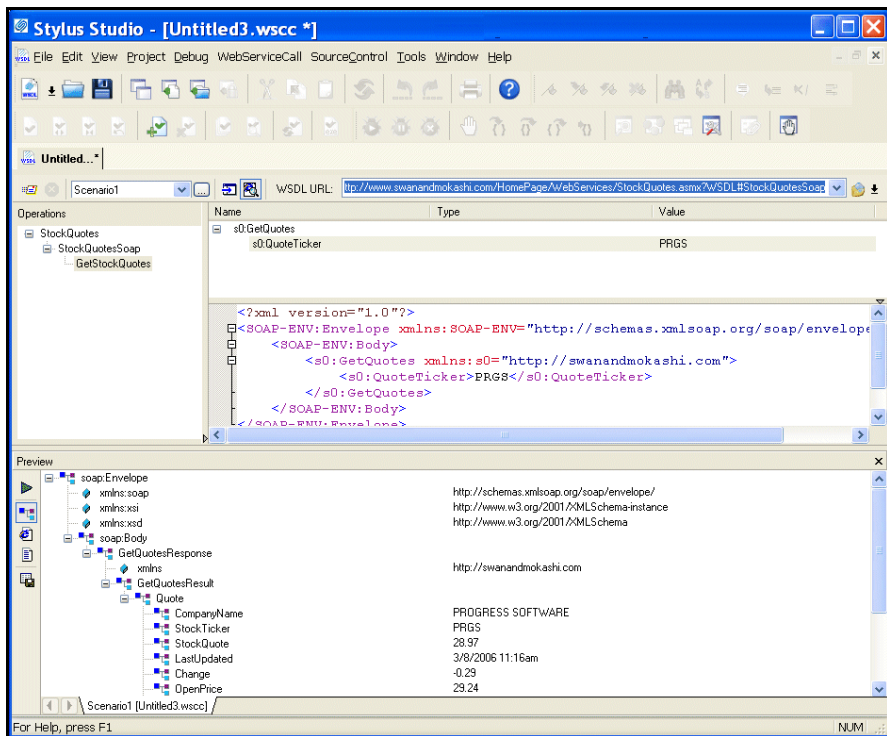


Figure 342. SOAP Response

6. Optionally, save the Web service call. See “[Saving a Web Service Call](#)” on page 839 for help with this step.

Obtaining WSDL URLs

Every Web service is described by a Web Services Description Language (WSDL). The WSDL defines the format of the SOAP messages used to send requests to and receive responses from the Web service, the transfer protocol used, namespace declarations, and other information. Several vendors, such as IBM, Microsoft, and SAP, have established Universal Description, Discovery, and Integration (UDDI) registries, to make Web services publicly available.

You can locate WSDLs on your own, or you can use Stylus Studio to search UDDI registries for published Web services.

Browsing UDDI Registries

You browse UDDI registries and search for published Web services (and the WSDLs that describe them) using the **UDDI Browser** dialog box.

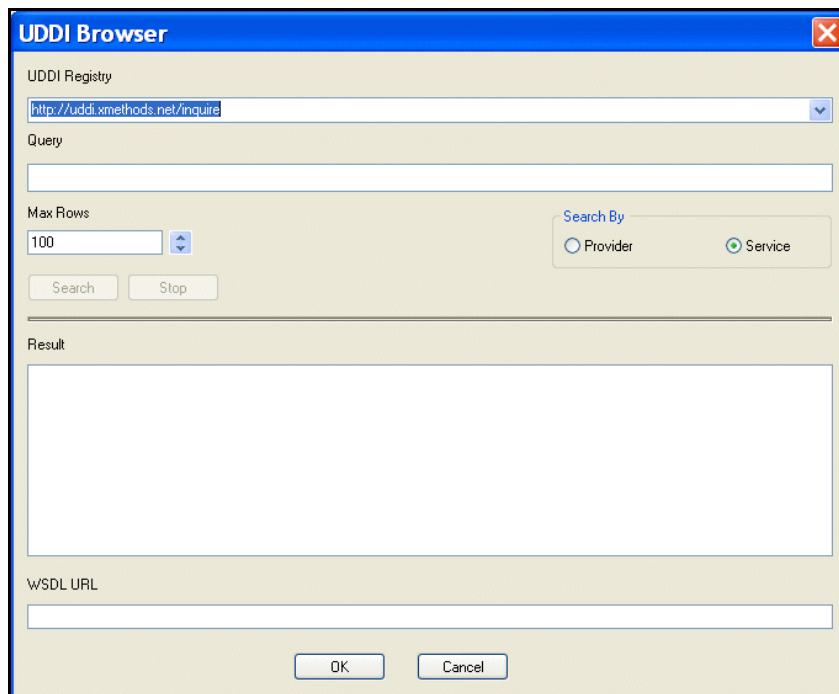


Figure 343. UDDI Browser Dialog Box

The **UDDI Registry** field displays a list of public UDDI registries. You use the **Query** field (obscured by the **UDDI Registry** drop-down list in the preceding illustration) to specify the keywords you want to use to search a UDDI registry from this list. For example, if you are building a weather application, you might type **weather** in the **Query** field to search for weather-related Web services. Keywords are matched against the Web service and company information available in the UDDI registry, not against the WSDL itself. Generally speaking, the same search executed against different UDDI registries will yield different results.

In addition to specifying keywords, the **UDDI Browser** dialog box allows you to

- Specify whether you want to search by Web service (the default) or by provider
- Limit the search results to a number or rows (the default is 100)

When you execute the search (by clicking the **Search** button), Stylus Studio displays search progress in a status bar. You can stop the search at any time by clicking the **Stop** button.

When the search is complete, the URLs for any WSDLs that meet your search criteria are displayed in the **Result** field. For example, if you search the XMethods UDDI registry for

Web services related to weather, the Stylus Studio **UDDI Browser** returns the following results:

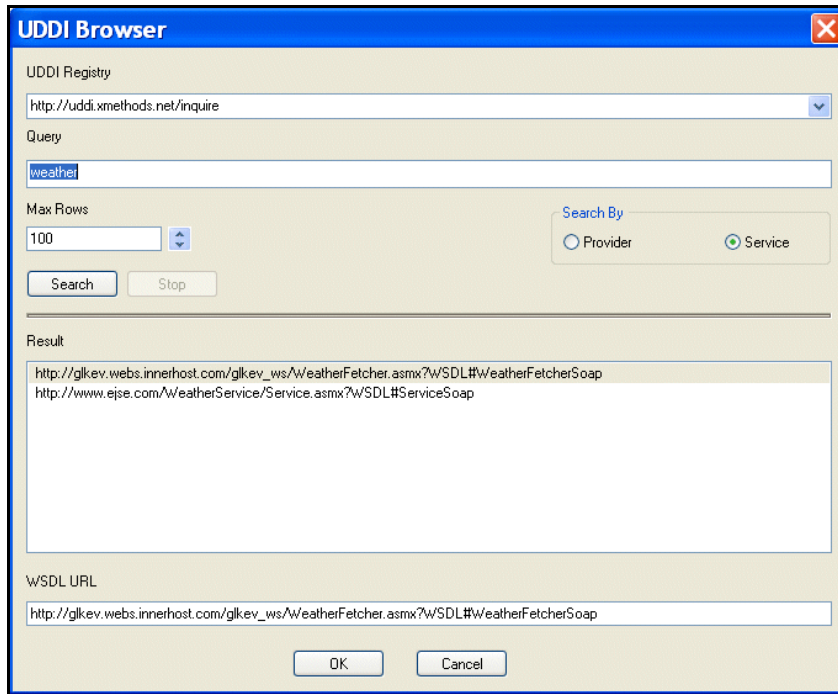


Figure 344. UDDI Browser result

When you select a WSDL URL, Stylus Studio displays the operations supported by the Web service in the Web Service Call Composer. The first operation is selected by default, and the SOAP request that defines it is displayed in the XML editing area. Web services can provide multiple operations. See [“Modifying a SOAP Request”](#) on page 835.

If you do not see a suitable WSDL URL in the UDDI registry you searched, modify your query in the **UDDI Browser** and try your search again, or search a different UDDI registry.

How to Browse UDDI Registries

◆ To browse UDDI registries:

1. In the Web Service Call Composer, click the **UDDI** button.
The **UDDI Browser** dialog box appears.

2. In the **UDDI Registry** field, type the URL of a UDDI registry, or select a UDDI registry from the drop-down list.
3. In the **Query** field, enter the string you want to use to search the selected UDDI registry for available Web services.
4. Optionally, change the following:
 - **Max Rows** – the maximum number of results you want displayed in the **Results** field.
 - **Search By** – whether you want to search the UDDI registry by company or by Web service (the default).
5. Click the **Search** button.

Search progress is displayed in a status bar. When the search is complete, WSDLs that match the search criteria you specified are displayed in the **Results** field.
6. Select the WSDL that defines the Web service operation for which you want to compose a SOAP request and click **OK**.

The **UDDI Browser** dialog box closes and you are returned to the Web Service Call Composer.

Modifying a SOAP Request

When you select a WSDL from the **Result** field in the **UDDI Browser** and click **OK**, the operations exposed by the Web service are displayed in the Stylus Studio Web Service Call Composer.

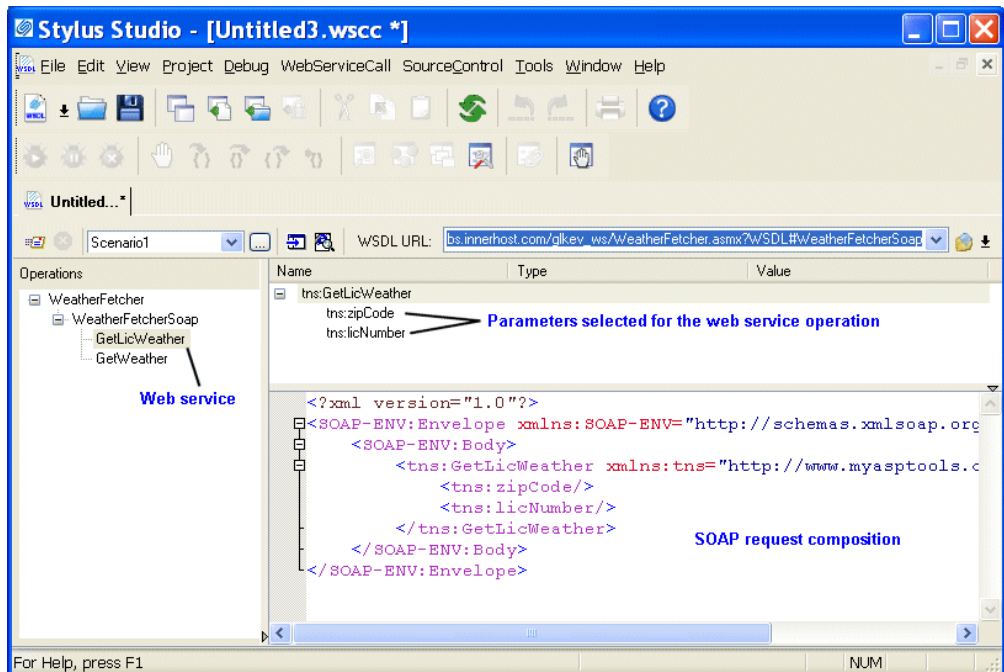


Figure 345. Modifying a SOAP request

When you select an operation from the **Operations** pane, Stylus Studio displays

- Parameters associated with that operation, including their datatype and a field in which you can enter a value for testing purposes.
- The XML that describes the SOAP request associated with that operation. You can edit the SOAP request, but for initial testing you should restrict changes to providing parameter values, and you can use the parameter's **Value** field for this purpose.

Understanding Parameters

Stylus Studio displays the datatype for SOAP request parameters. It is not possible to determine all of the details for parameters, however. A zipCode parameter might take the following:

12309, 02134, 90210

Or it might take only a single value. Sometimes this type of information is provided in the WSDL itself. In some cases, however, you might have to contact the Web service provider to obtain this information.

Displaying a WSDL Document

You can easily display a WSDL document within Stylus Studio once you have specified the WSDL URL. You might want to look at a WSDL document to learn more about the structure of the SOAP request, or to see if the Web service provider commented the XML to include information for developers using their Web service.

How to display a WSDL document

- ◆ **To display a WSDL document, click the Open WSDL Document button () near the top of the Web Service Call Composer.**

Stylus Studio displays the WSDL document in its own XML editor, as shown in [Figure 346](#):

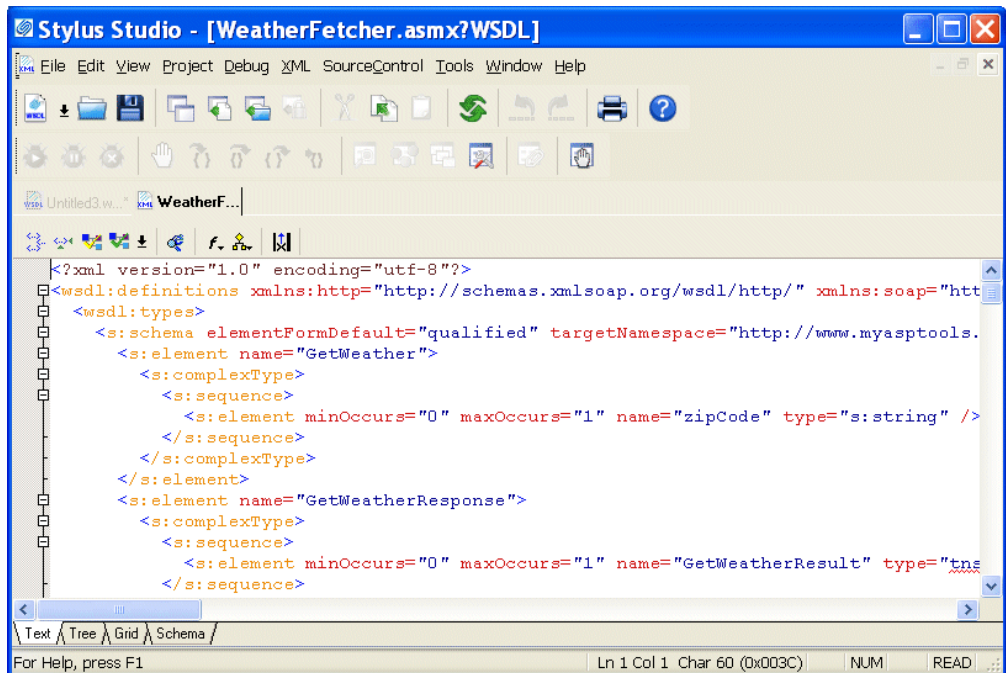


Figure 346. WSDL document editor

How to Modify a SOAP Request

◆ To modify a SOAP request:

1. Select the Web service operation for which you want to compose a SOAP request from the **Operations** pane.
The SOAP request for the Web service operation appears in the XML editing area.
2. In the **Name** field, select a parameter and enter a value for it in the **Value** field.
3. Repeat [Step 2](#) for any remaining parameters.

Once you have specified values for the SOAP request's parameters, you can test the Web service. See [“Testing a Web Service”](#) on page 837.

Testing a Web Service

You can test a Web service from within Stylus Studio. Testing allows you to quickly and easily

- Verify whether or not the Web service is available
- Understand whether or not the Web service provides the type of information you expect and require
- Learn about the SOAP response returned by the Web service
- Learn how parameters you might choose to specify in a Web service call scenario affect the Web service operation

What Happens When You Test a Web Service

When you test a Web service, Stylus Studio submits the SOAP request to the WSDL URL specified in the Web service call. The result, when it is returned, is displayed in the **Preview** window of the Web Service Call Composer.

By default, Stylus Studio uses the HTTP transport protocol to submit the SOAP request to the WSDL server. Stylus Studio uses the proxy server specified on the local machine if one has been configured.

Other Options for Testing a Web Service

In addition to testing a Web service as described in this section, you can also create a Web service call scenario. Web service call scenarios allow you to

Composing Web Service Calls

- Use transport protocols besides HTTP
- Specify overrides to the WSDL (changing the SOAP action, for example)
- Change default settings (such as the time out value for executing SOAP requests)

See “[Creating a Web Service Call Scenario](#)” on page 846 to learn more about Web service call scenarios.

How to Test a Web Service

- ◆ **To test a Web service, click the Send Request button () to submit the SOAP request.**

Stylus Studio displays the SOAP response in the **Preview** window as shown in [Figure 347](#):

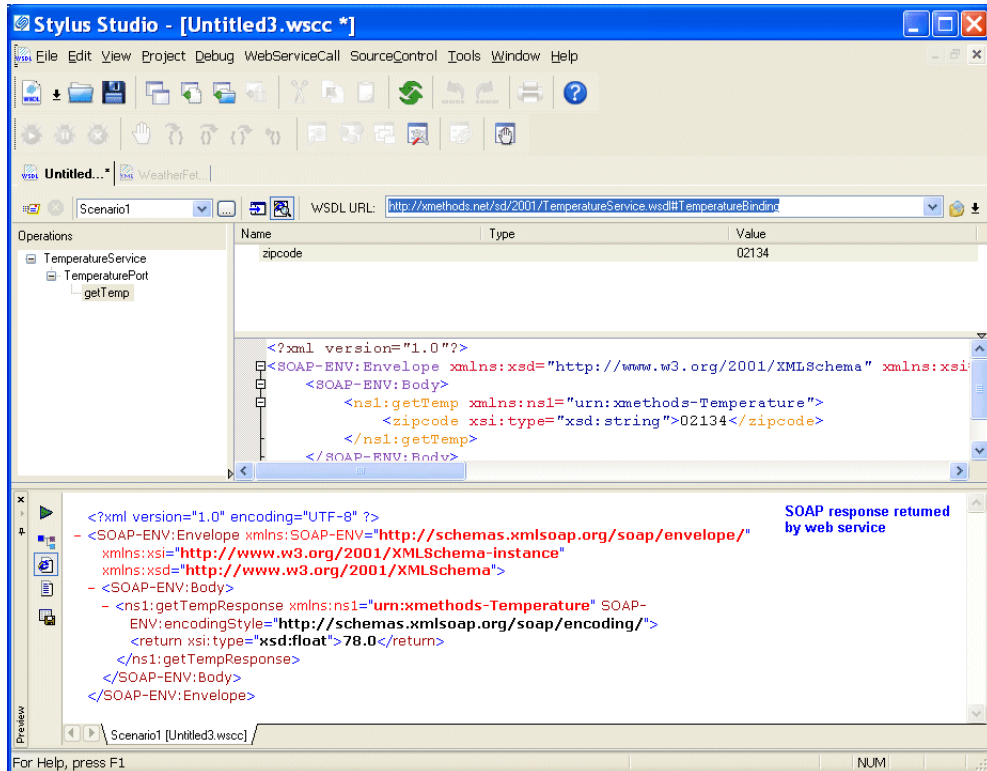


Figure 347. Testing a Web service

Saving a Web Service Call

You can save the Web service call composed by Stylus Studio. The file created when you save a Web service call includes the WSDL URL and the last Web service operation that you configured (including parameter values) prior to saving the file. For example, if you are using a Web service that provides separate operations for temperature conversions (one for Celsius to Fahrenheit and one for Fahrenheit to Celsius, for example), only the last one you test is saved.

Saving a Web service call gives you the ability to easily recall a preconfigured SOAP request for additional testing – allowing you to modify the SOAP request and test it without having to locate the WSDL.

Using Web Service Calls as XML

In addition to opening a Web service call in the Web Service Call Composer for testing purposes, you can open a Web service call as an XML document anywhere in Stylus Studio – in the XML editor, or as a source document in the XQuery mapper for example. When you open a Web service call as an XML document, Stylus Studio automatically executes the SOAP request and displays the SOAP response.

Consider the following Web service call, `stock.wsc`. The Web service operation used in this example returns current stock quote and other information based on the ticker symbols provided as parameters. Here is the SOAP request composed by Stylus Studio:

```
<?xml version="1.0" standalone="no"?>
<SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  SOAP-ENV:Body>
    <s0:GetStockQuotes xmlns:s0="http://swanandmokashi.com/">
      <s0:QuoteTicker>prgs</s0:QuoteTicker>
    </s0:GetStockQuotes>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
<soap:Body>
  <GetStockQuotesResponse xmlns="http://swanandmokashi.com/">
    <GetStockQuotesResult>
      <Quote>
        <CompanyName>PROGRESS SOFT</CompanyName>
        <StockTicker>PRGS</StockTicker>
        <StockQuote>20.10</StockQuote>
        <LastUpdated>10:17am</LastUpdated>
        <Change>+0.03</Change>
        <OpenPrice>20.05</OpenPrice>
        <DayHighPrice>20.40</DayHighPrice>
        <DayLowPrice>20.00</DayLowPrice>
        <Volume>13200</Volume>
        <MarketCap>695.1M</MarketCap>
        <YearRange>11.50 - 24.06</YearRange>
      </Quote>
    </GetStockQuotesResult>
  </GetStockQuotesResponse>
</soap:Body>
</soap:Envelope>
```

And here is the SOAP response returned by the Web service:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

The saved Web service call can be used as the source document for an XQuery in the XQuery mapper, as shown in [Figure 348](#):

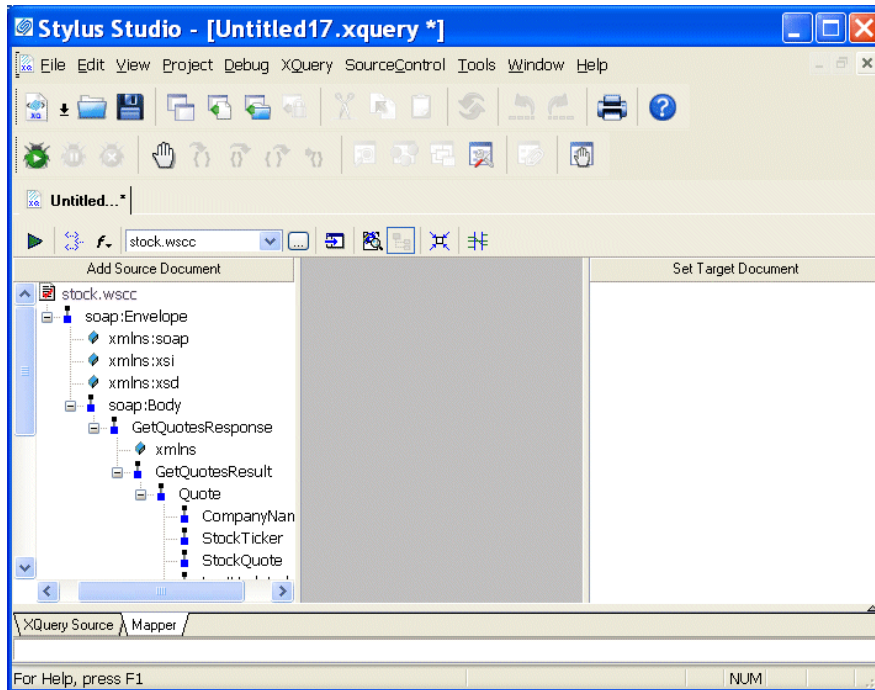


Figure 348. Using a Web Service Call to Compose an XQuery

XQueries composed using a Web service call as a source document return real-time data from the Web service as a result.

How to Save a Web Service Call

◆ To save a Web service call:

1. Select **File > Save** from the Stylus Studio menu bar.
The first time you save a Web service call, the **Save As** dialog box appears; for subsequent save operations, Stylus Studio displays the **Save** dialog box.
2. Change the default name (Untitled.wsc, for example), and click **Save**.

Generating a Java Web Service Client

You can use Stylus Studio to generate Java classes based on the currently active Web service call. You can then use those classes to write a Java application that allows you to invoke that Web service programmatically.

About the Generated Code

This section describes details about the Java code generated from a Web service call.

The Package Name

Stylus Studio bases the package name on the target namespace of the WSDL. For example, if we use a WSDL from `www.swanandmokashi.com`, the resulting Java package name is `com.shawandmokashi`. As described in the following sections, target directory and project folder names are also based on the WSDL's target namespace.

Target Directory

By default, when you generate a Java Web service client, Stylus Studio creates a `\java` folder in your system's `\temp` directory (`c:\temp\java`, for example). Subfolders based on the package name are created beneath the `\java` folder, and the Java classes are written there. For example, `c:\temp\java\com\swanandmokashi*.java`.

You can specify whatever target directory you like, but the subfolders based on the package name cannot be specified when you generate the Java code.

Project Folders

By default, Stylus Studio adds the Java classes based on the Web service call to the current project, using the name of the target directory you specify; subfolders are based on the package name (`\java\com\swanandmokashi`, for example).

Classes

Stylus Studio generates a separate Java class for each function described in the WSDL of the current Web service call. Using the `StockQuotes` Web service from `www.swanandmokashi.com`, for example, results in the generation of the following Java classes:

- `GetQuotes.java`

- GetQuotesResponse.java
- Quote.java
- StockQuotes.java
- StockQuotesLocator.java
- StockQuotesSoap.java
- StockQuotesSoapStub.java

How to Generate a Java Web Service Client

◆ To save a Web service call:

1. Create a Web service call as described in “How to Compose a Web Service Call” on page 828.
2. Test the SOAP request by clicking the **Preview Result** button, and verify that it returns the results you require.
3. Click **WebServiceCall > Generate Java Web Service Client** from the Stylus Studio menu.

The Generate Java Web Service Client dialog box appears.

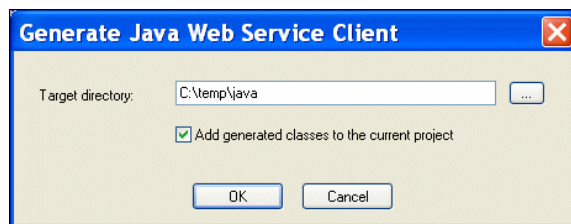


Figure 349. Generate Java Web Service Client Dialog Box

4. Optionally, change the default target directory.
5. Optionally, change the default setting to add generated classes to the current Stylus Studio project.
6. Click OK.
Stylus Studio generates the Java classes for the WSDL in the current Web service call.

Generating XQuery from a Web Service Call

You can use Stylus Studio to generate XQuery from a Web service call, and then use that XQuery to invoke the Web service. The XQuery created by Stylus Studio uses a Java extension function, `ws:call()`, that allows the built-in, Saxon, or DataDirect XQuery® processor to execute the Web service call.

This section covers the following topics:

- [“Example”](#) on page 844
- [“What Happens When You Generate XQuery”](#) on page 845
- [“How to Generate XQuery from a Web Service Call”](#) on page 845

Example

Consider the following StockQuotesSoap SOAP request defined using the Swan and Mokashi WSDL:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <s0:GetQuotes xmlns:s0="http://swanandmokashi.com">
      <s0:QuoteTicker>prgs</s0:QuoteTicker>
    </s0:GetQuotes>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This SOAP request was created using the Stylus Studio Web Service Call Composer, as described in [“How to Compose a Web Service Call”](#) on page 828.

The XQuery generated by Stylus Studio for this Web service call looks like this:

```
declare namespace ws = "ddtekjava:com.stylusstudio.webservice.SOAPCall";
declare function ws:call($location as element(), $payload as element()) as
document-node() external;

ws:call(
  <location
    address="http://www.swanandmokashi.com/HomePage/WebServices/StockQuotes.asmx"
    soapaction="http://swanandmokashi.com/GetQuotes" />,
  <s0:GetQuotes xmlns:s0="http://swanandmokashi.com">
    <s0:QuoteTicker>prgs</s0:QuoteTicker>
  </s0:GetQuotes>
)
```

In addition to the header that declares the `ws` namespace and Java extension function, `ws:call()`, the generated code includes comments that specify `.jar` files that must be present

in your Classpath in order to run the resulting XQuery in Stylus Studio. Here's an example of the complete code generated by Stylus Studio for a Swan & Mokashi Web service:

```
(: This function requires the following items in the classpath :)
(: axis-all.jar;xercesimpl.jar :)
(: xml-apis.jar only if running on Java 1.4 :)

(: Saxon function binding :)
(:declare namespace ws = "java:com.stylusstudio.webservice.SOAPCall";:)

(: DataDirect XQuery function binding :)
declare namespace ws = "dtekjava:com.stylusstudio.webservice.SOAPCall";
declare function ws:call($location as element(), $payload as element()) as
document-node() external;

ws:call(
  <location
address="http://www.swanandmokashi.com/HomePage/WebServices/StockQuotes.asmx"
soapaction="http://swanandmokashi.com/GetQuotes" />,
  <s0:GetQuotes xmlns:s0="http://swanandmokashi.com">
    <s0:QuoteTicker>prgs</s0:QuoteTicker>
  </s0:GetQuotes>
)
```

What Happens When You Generate XQuery

When you generate XQuery from a Web service call, Stylus Studio copies the resulting XQuery to your system's clipboard. From there, you can paste it into a new or existing XQuery.

See [“Using XQuery to Invoke a Web Service”](#) on page 816 for more information on working with XQuery generated from a Web service call.

How to Generate XQuery from a Web Service Call

◆ To generate XQuery from a Web service call:

1. Create a Web service call as described in [“How to Compose a Web Service Call”](#) on page 828.
2. Test the SOAP request by clicking the **Preview Result** button, and verify that it returns the results you require.
3. Click **WebServiceCall > Copy XQuery to Clipboard** on the Stylus Studio menu. Stylus Studio creates an XQuery based on the SOAP request and copies the generated XQuery code to your system's clipboard.

Creating a Web Service Call Scenario

A *Web service call scenario* is a group of customizable settings associated with a Web service call composition. Stylus Studio uses these settings when you test a Web service using a scenario. If you don't define a scenario, or don't test the Web service call using a scenario, Stylus Studio uses the settings described in the WSDL. Examples of Web service call scenario settings include the client used to perform the Web service call; a username and password for Web services requiring authentication; and the length of time Stylus Studio will try to access the Web service before timing out.

You should consider creating a Web service call scenario only after you have defined the Web service call itself. This allows Stylus Studio to inherit values for the scenario from the WSDL you select for your Web service call.

You can create multiple scenarios that use the same Web service call, and define different settings for each. This flexibility can aid the Web service call development process as it enables you to easily test different Web service parameters before making the Web service call available in your XML applications. A scenario can be associated with only one Web service call.

This section covers the following topics:

- [“Overview of Scenario Features”](#) on page 846
- [“How to Create a Scenario”](#) on page 849
- [“How to Run a Scenario”](#) on page 850
- [“How to Clone a Scenario”](#) on page 850

Overview of Scenario Features

This section describes the main features of Web service call scenarios. It covers the following topics:

- [“Scenario Names”](#) on page 847
- [“Transport Protocol and Client Settings”](#) on page 847
- [“Other Transport Settings”](#) on page 847

Scenario Names

You specify a name for a Web service call scenario on the **Binding** tab of the **Scenario Properties** dialog box.

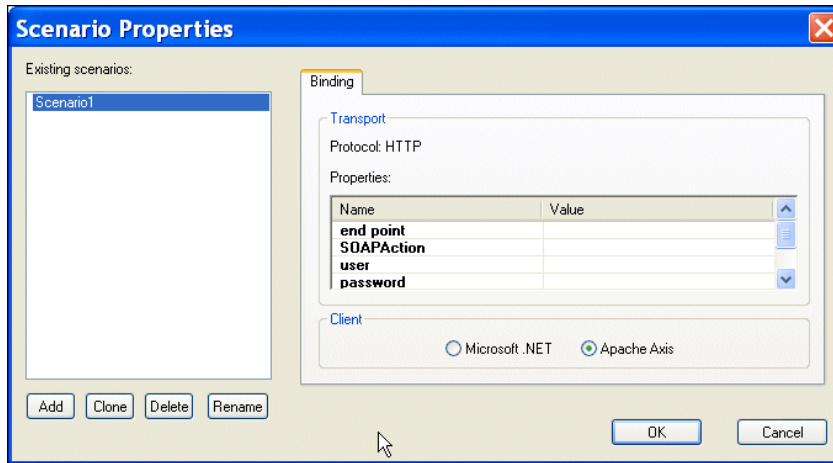


Figure 350. Binding Tab of the Web Service Call Scenario Properties Dialog Box

When you create a Web service call scenario, specify a name that makes it easy to distinguish one scenario from another.

Transport Protocol and Client Settings

You specify the transport protocol you want to use when testing the Web service on the **Binding** tab of the **Scenario Properties** dialog box.

When you use HTTP as the transport protocol, the Web service call client can be any one of the following:

- Microsoft .NET
- Apache Axis

Other Transport Settings

Once you specify the client, Stylus Studio displays a list of additional settings that you can use to define properties for the scenario. Some values, such as the time out, are system

defaults. Others, such as the SOAP action, are taken directly from the WSDL specified in the Web Service Call Composer.

Note Values you specify on the **Binding** tab override those in the WSDL displayed in the Web Service Call Composer.

HTTP Settings


The following table describes the scenario settings associated with the HTTP transport protocol.

Table 99. HTTP Settings

Setting	Description
end point	The server on which the Web service is executed. For example: <code>http://glkev.webs.innerhost.com/glkev_ws/WeatherFetcher.asmx</code> . This value is taken from the current Web service call. Required.
SOAPAction	The SOAP action described by the WSDL you selected for the Web service call. For example: <code>http://www.myasptools.com/GetWeather</code> This value is taken from the current Web service call. Required.
user	The username used to access the Web service if authentication is required. Optional.
password	The password used to access the Web service if authentication is required. Optional.
time out	The time in milliseconds until the connection to the Web service server is dropped due to inactivity. The default is 300000 (300 seconds). Required.

How to Create a Scenario

◆ **To create a scenario:**

1. Create a Web service call if you haven't already. See [“How to Compose a Web Service Call”](#) on page 828 if you need help with this step.
2. Display the **Scenario Properties** dialog box by clicking  in the Web service editor tool bar.

Alternative: Select **Create Scenario** from the scenario drop-down list at the top of the editor window:

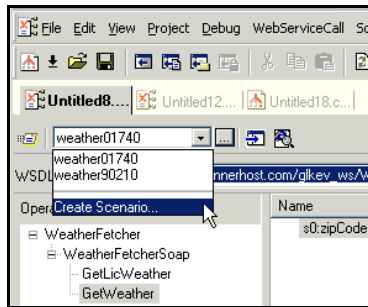


Figure 351. Creating a Scenario

3. On the **General** tab, specify a name for the Web service call scenario.
4. Click the **Binding** tab.
5. Select the appropriate transport protocol from the **Transport** drop-down list.
6. Specify the binding properties you want to associate with this Web service call scenario.
7. Click **OK**.

The Web service call scenario is saved with the name and settings you specified.

How to Run a Scenario

◆ **To run a scenario:**

1. Select a scenario from the scenario drop-down list at the top of the editor window:

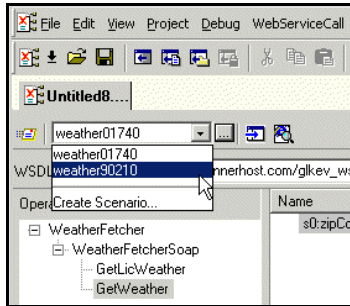




Figure 352. Picking a Saved Scenario


Alternative:

- a. In the Web Service Call Composer tool bar, click . Stylus Studio displays the **Scenario Properties** dialog box.
 - b. On the **General** tab, select the scenario you want to run from the **Existing preview scenarios** list.
 - c. Click **OK**.
2. Click the **Send Request** button () .

How to Clone a Scenario

When you clone a scenario, Stylus Studio creates a copy of the scenario except for the scenario name. This allows you to make changes to one scenario and then run both to compare the results.

◆ **To clone a scenario:**

1. Display the **Scenario Properties** dialog box by clicking  in the Web Service Call Composer tool bar.
2. In the **Existing preview scenarios** field, click the name of the scenario you want to clone.
3. Click **Clone**.

4. In the **Scenario name** field, type the name of the new scenario.
5. Change any other scenario properties you want to change. See [Overview of Scenario Features](#) on page 846.
6. Click **OK**.
If you change your mind and do not want to create the clone, click **Delete** and then **OK**.

Chapter 12 Building XML Pipelines

This chapter describes XML pipelines, and how to use the Stylus Studio XML Pipeline Editor to create, debug, and maintain XML pipelines. It also describes how to generate Java code you can use to embed XML pipelines in Java applications.



Support for XML pipelines is available only in Stylus Studio XML Enterprise Suite.



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the Introduction to the XML Pipeline Editor video](#).

A complete list of the videos demonstrating Stylus Studio's features is here:
http://www.stylusstudio.com/xml_videos.html.

This chapter covers the following topics:

- “[What is an XML Pipeline?](#)” on page 854
- “[The XML Pipeline Editor](#)” on page 857
- “[Steps for Building an XML Pipeline](#)” on page 860
- “[Planning an XML Pipeline](#)” on page 861
- “[Use Case: Building order.pipeline](#)” on page 867
- “[Working with Nodes](#)” on page 893
- “[Working with the XML Pipeline Diagram](#)” on page 911
- “[Debugging an XML Pipeline](#)” on page 916
- “[Generating Code for an XML Pipeline](#)” on page 920
- “[XML Pipeline Node Properties Reference](#)” on page 924

What is an XML Pipeline?

In Stylus Studio, an *XML pipeline* is an application that performs a series of operations based on the inputs, transformations, and outputs described in the XML Pipeline Editor. In Stylus Studio, an XML pipeline has a

- Graphical representation consisting of nodes that represent data sources, processing operations, and pipes that represent the processing flow (shown in [Figure 353](#))
- A code representation (once you generate code for it) for these data sources, nodes, and pipes

Example of an XML Pipeline in Stylus Studio

[Figure 353](#) shows the diagram that represents `getHoldings.pipeline`, which is in the `pipelines\stocks` folder in the `examples` project installed with Stylus Studio.

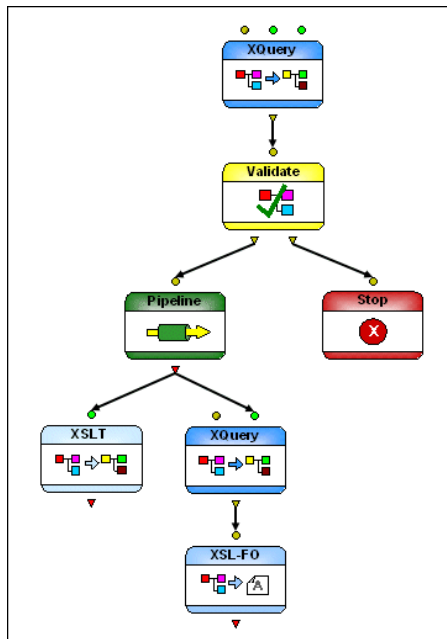


Figure 353. Diagram of `getHoldings.pipeline`

This particular XML pipeline:

- Aggregates two XML input sources
- Validates the output of the XQuery using XML Schema and then either

- Terminates if the validation fails or
- Passes the output to an embedded pipeline for additional processing
- Using XSLT, transforms the embedded pipeline's output to HTML
- Using XQuery, transforms the same output to PDF using XSL-FO processing

XML Pipeline Terminology

Understanding the following terms will help you work with XML pipelines in Stylus Studio.

Table 100. XML Pipeline Terminology

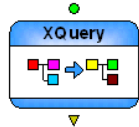
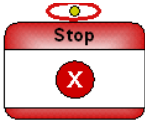
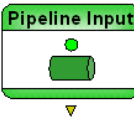
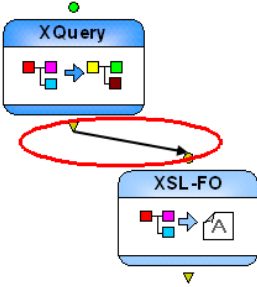
<i>Term</i>	<i>Description</i>	<i>Example</i>
Node	Generally, an XML pipeline operation. In an XML pipeline diagram, the glyph that represents an XML pipeline operation. Examples include XSLT, XQuery, Pipeline, and Validate.	 A blue rounded rectangle with a white border, labeled "XQuery" in black text. Inside, there are several small colored squares (red, blue, green, yellow) and a white arrow pointing right. A green circle is at the top center, and a yellow triangle points down at the bottom center.
Input port	The circle on top of some nodes used to receive a pipe from an output port. A node can have more than one input port.	 A red rounded rectangle with a white border, labeled "Stop" in black text. Inside, there is a red circle with a white "X" in the center. A red circle with a white dot is at the top center, and a red circle with a white "X" is at the bottom center.

Table 100. XML Pipeline Terminology

<i>Term</i>	<i>Description</i>	<i>Example</i>
Output port	The triangle on the bottom of some nodes used to connect to an input port using a pipe. A node can have more than one output port.	
Pipe	The conceptual name for the line that connects two nodes, from the output port on one, to the input port on the other.	

XML Pipeline Semantics

The following semantics govern the behavior of an XML pipeline:

- A node is executed only when all its input ports are “filled”.
- A node’s input port is “filled” either when it contains a default value that is a reference to a URL or a literal value, or when it is connected to another node’s output port and this node provides data that is available. Default values are used only if no pipe is present for that port.
- When data is available on a node’s output port, it is provided to the input ports of all the nodes to which it is connected.
- A node’s input port can be filled by 0 or 1 value. If more than one value becomes available, this is an error and XML pipeline processing aborts.

The XML Pipeline Editor

The *XML Pipeline Editor* is the visual editing tool you use to create, execute, and debug XML pipelines in Stylus Studio.

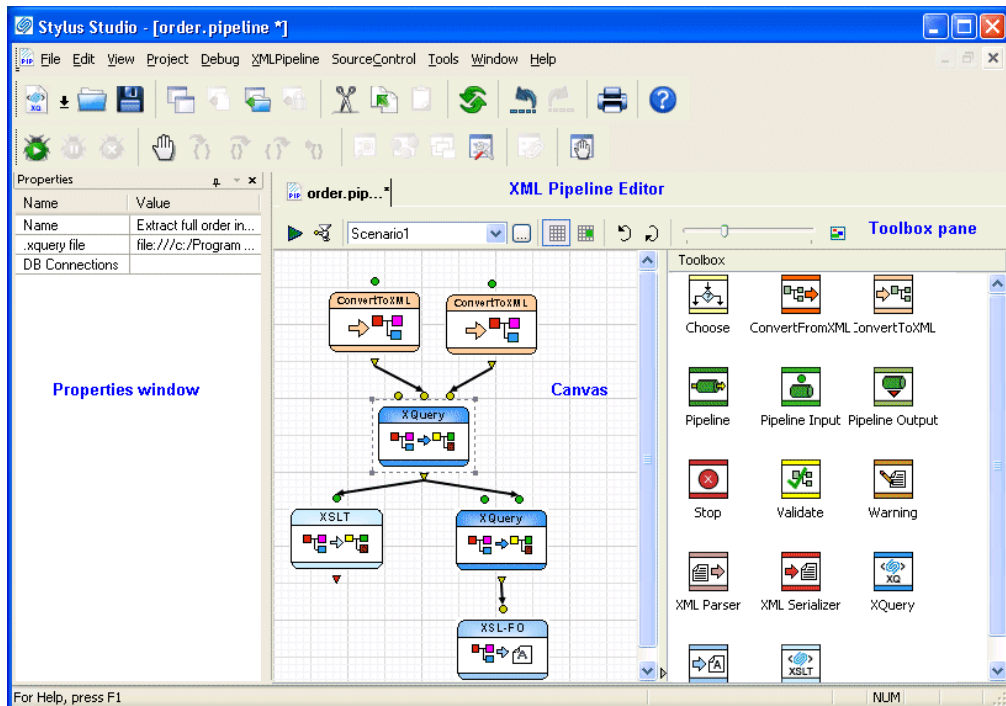


Figure 354. Example of an XML Pipeline in the XML Pipeline Editor

You build an XML pipeline in the *XML Pipeline Editor* using simple drag-and-drop actions – to add an XQuery query to an XML pipeline, for example, you can drag an XQuery file from the **Project** window and drop it on the XML Pipeline Editor canvas. You can also use tools from the **Toolbox** pane to add nodes to your XML Pipeline. The **Properties** window allows you to specify settings for the node (default input and output values, for example).

You can also use the XML Pipeline Editor to

- Execute the XML pipeline and preview its output
- Generate Java code for an XML pipeline
- Create an image of the XML pipeline diagram

This section covers the following topics:

- [“Parts of the XML Pipeline Editor”](#) on page 858
- [“XML Pipeline Editor Toolbar”](#) on page 859
- [“Menu Actions”](#) on page 860

Parts of the XML Pipeline Editor

The XML Pipeline Editor has three main parts, shown in [Figure 354](#):

- The *XML pipeline canvas*, on which you compose the operations and the flow of your XML pipeline and work with the resulting XML pipeline diagram. See [“Working with the XML Pipeline Diagram”](#) on page 911 for more information.
- The *Toolbox pane* contains the tools you use to add transformation, flow control, and data source operations to your XML pipeline. Operations you add using toolbox tools are not yet tied to an implementation – you need to specify them by setting values in the **Properties** window. See [“Adding Nodes to an XML Pipeline”](#) on page 893 for more information.
- The *Properties window*, which displays information about the operations in your XML pipeline. Some default values are provided by Stylus Studio; if you create an operation by dragging a document from the **Project** window and dropping it on the canvas, however, some of that document’s information is used to specify property settings for the operation. For example, if you drag and drop an XQuery document in the XML pipeline, the **.xquery file** property is based on the document’s URL. See [“XML Pipeline Node Properties Reference”](#) on page 924 for more information on properties for individual nodes.

XML Pipeline Editor Toolbar

The XML Pipeline Editor toolbar provides easy access to operations you are likely to perform while building XML pipelines and working with XML pipeline diagrams in Stylus Studio. [Figure 355](#) identifies the toolbar's tools.

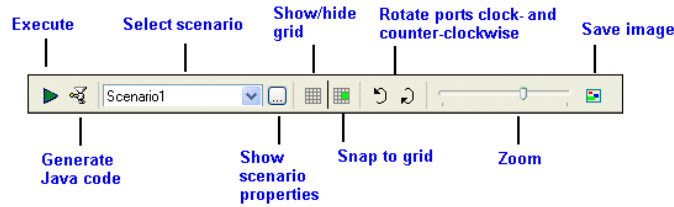


Figure 355. XML Pipeline Editor Toolbar Buttons

Many of these operations are also available from the **XMLPipeline** menu and canvas short-cut menu (right-click to display). See [“Menu Actions”](#) on page 860 for more information.

Table 101. Toolbar Button Descriptions

<i>Toolbar Button</i>	<i>Description</i>
Execute	Executes the current XML pipeline and displays the results in the Preview window. Output (error and warning messages, for example) appear in the Output window.
Generate Java Code	Generates Java code for the current XML pipeline.
Select Scenario	Allows you to choose the XML pipeline scenario for execution and code generation.
Display Scenario Properties	Displays the Scenario Properties dialog box, which allows you to create and define properties for XML pipeline scenarios.
Show Grid	Shows (or hides) the grid that appears on the XML Pipeline Editor canvas.
Snap to Grid	Allows you to choose whether you want objects to be placed automatically on the closest grid line (snap), or whether you want to be able to place them anywhere on the grid you choose. Snap to grid is off by default.

Table 101. Toolbar Button Descriptions

<i>Toolbar Button</i>	<i>Description</i>
Rotate Ports Counter-Clockwise	Allows you to rotate the ports on the selected node counter-clockwise.
Rotate Ports Clockwise	Allows you to rotate the ports on the selected node clockwise.
Zoom	Zooms the XML Pipeline diagram on the canvas.
Save Image	Saves the entire XML Pipeline diagram (not just what is visible on the canvas) to an image file.

Menu Actions

In addition to the actions you can perform using the toolbar (see “[XML Pipeline Editor Toolbar](#)” on page 859), the **XMLPipeline** menu and canvas short-cut menu provide the following actions.

Table 102. Menu Action Descriptions

<i>Menu Action</i>	<i>Description</i>
Add Label/Remove Label	Allows you to add (and remove) labels from the XML pipeline diagram.
Edges Style	Lets you choose line, orthogonal, or spline styles for the pipes in the XML pipeline diagram.
Remove Selection	Removes the selected node or pipe from the XML pipeline.

Steps for Building an XML Pipeline

The process for building an XML pipeline consists of these basic steps:

1. Gather the requirements for the XML pipeline and select a design approach. You need to understand the XML pipeline’s desired output, and perhaps which XML technologies (XQuery or XSLT, for example) need to be used. See “[Planning an XML Pipeline](#)” on page 861.
2. Identify and/or define the source documents and resources required to execute your XML pipeline.

3. Create an XML pipeline document. See [“Getting Started: Creating a New XML Pipeline”](#) on page 869.
4. Optionally, set deployment and processor properties. See [“XML Pipeline Scenarios”](#) on page 870.

Note You can do this step any time prior to previewing and debugging the XML pipeline.

5. Create and specify the XML pipeline nodes. You can create empty nodes and specify them manually, or you can create them using XQuery, XSLT, XML Schema, and other documents. See [“Working with Nodes”](#) on page 893.
6. Use pipes to connect the nodes in your XML pipeline.
7. Test the XML pipeline. See [“Testing the XML Pipeline”](#) on page 883.
8. Debug the XML pipeline as needed. See [“Debugging an XML Pipeline”](#) on page 916.

Once you are satisfied that the XML pipeline is running as it should, you can optionally generate Java code. See [“Generating Code for an XML Pipeline”](#) on page 920 for more information.

Planning an XML Pipeline

This section describes some of the considerations you might want to make when planning an XML pipeline – which design approach to take, the components you can include in a pipeline, and more.

This section covers the following topics:

- [“Design Approaches”](#) on page 861
- [“XML Pipeline Components”](#) on page 864
- [“Identifying Resources”](#) on page 866
- [“Deployment Considerations”](#) on page 867
- [“Steps for Building an XML Pipeline”](#) on page 860

Design Approaches

Stylus Studio supports bottom-up and top-down approaches to designing XML pipelines. The approach you take depends largely on personal preference, but it can also be influenced by factors such as whether, for example, your XML pipeline will use existing

transformations (like XQuery or XSLT) or you will build them specifically for use in the XML pipeline.

The following section is intended to give you some ideas for XML pipeline design.

Understand the Requirements

Regardless of which approach you choose, you should understand the goal of the XML pipeline before you start building it. For example, you should know

- What the desired output is. Is it HTML? XSL-FO? Both? Or will the XML pipeline return data to a format other than XML?
- If the XML pipeline is intended to stand alone, or whether it will be included in other XML pipelines.

For the purposes of describing bottom-up and top-down design approaches in this section, imagine that the requirement for our XML pipeline is to reder data in a text file as PDF.

Bottom-Up Design

In a bottom-up design approach, you already have the individual components, or most of them, that you will link together to form your XML pipeline. If we were using a bottom-up design approach to create an XML pipeline for the use case described in [“Understand the Requirements”](#) on page 862, we would:

- Have a source .txt file (a comma-separated values file) identified.
- Use a built-in Stylus Studio converter to convert this file to XML.
- Have an XQuery file that transforms this XML to XSL-FO and performs FO post-processing to create PDF.

Using a bottom-up design approach, we would then use these source files to build our XML pipeline using the following steps:

1. Create a new XML pipeline document.
2. Create a ConvertToXML node to handle the conversion of CSV to XML. We would specify the source .txt file as the ConvertToXML node’s input, and choose the Comma-Separated Values built-in converter to convert the text to XML.
3. Drag our XQuery document from the **File Explorer** or **Project** window and drop it on the XML pipeline canvas. This would automatically create and specify the XML pipeline’s XQuery node for our XML pipeline. Also, because the XQuery was defined to perform XSL-FO post-processing, Stylus Studio automatically would create an XSL-FO node in the XML pipeline.

4. Connect the output port from the ConvertToXML node to the input port of the XQuery node. This instructs the XML pipeline to use the converted text file (now XML) as input for the XQuery transformation.
5. Specify a URL for the XSL-FO node's output port.

When you use a bottom-up approach, Stylus Studio leverages as much of the existing information in the documents you use to build the XML pipeline as possible. Depending on your design environment, you might need to alter the paths specified for input and output nodes, source documents, and so on, and you will typically have to link the nodes in your XML pipeline by creating pipes between appropriate output and input nodes.

Top-Down Design

When you use a top-down design approach, you do not have any pre-existing components – XSLT or XQuery documents, for example – or they might not be completely specified. In this situation, you use the XML Pipeline Editor to sketch a design, and then fill in the details once you have them. Returning to the use case described in [“Understand the Requirements”](#) on page 862, we would sketch our XML pipeline by:

1. Creating a new XML pipeline document.
2. Dragging a ConvertToXML icon from the **Toolbox** pane and dropping it on the canvas.
3. Dragging an XQuery icon from the **Toolbox** pane and dropping it on the canvas.
4. Dragging an XSL-FO icon from the **Toolbox** pane and dropping it on the canvas.
5. Connecting the ConvertToXML's output port to the input port of the XQuery node.
6. Connecting the XQuery node's output port to the input port of the XSL-FO node.

The top-down approach results in a rough outline of placeholder nodes of the desired XML pipeline – an abstract or conceptual representation of the code we want to generate to perform XML processing. The next steps would be to:

1. Identify the source document for the ConvertToXML node, and selecting the built-in Stylus Studio converter to be used to convert that source file's data to XML.
2. Creating an XQuery document that
 - Transforms the XML input from the ConvertToXML node to PostScript
 - Generates the XSL-FO grammar to convert the PostScript to PDF

Once these documents are created, they can be used to define nodes that represent them in the XML pipeline. You can do this

- Manually, by specifying node properties in the **Properties** window
- Automatically, by dragging and dropping documents onto the placeholder nodes that represent them

XML Pipeline Components

Every XML pipeline consists of a number of components that represent some aspect of XML processing. Typically, an XML pipeline will contain components that represent

- XML transformations (such as XQuery or XSLT)
- Source documents and data (an XML, or XML data provided by a Web service, for example)
- A flow that identifies the processing stages performed in the XML pipeline (whether XML output goes directly to an XSLT transformation for processing, or is first validated using XML Schema, for example)

You also specify values for the input and output ports on these nodes, which determines the flow of the processing defined in the XML pipeline.

This section reviews the components you can include in an XML pipeline.

Transformations

A *transformation* is an operation that takes an input, performs an action on it, and returns an output. Examples of XML transformations include XQuery and XSLT. Transformation output can be a finished product – XSLT that creates an HTML report, for example – or it can be something that is passed along to another operation for additional processing – XQuery that specifies FO post-processing of the XML it generates, or output passed to an XML Schema for validation, for example.

You can include the following transformations in an XML pipeline in Stylus Studio:

- XQuery – standard XQuery query, including scenario properties
- XSLT – standard XSLT transformation, including scenario properties
- XSL-FO – XSL-FO processing of XML using Apache FOP or RenderX XEP
- Pipeline – include one pipeline in another

- XML Parser – converts text input to XML
- XML Serializer – converts XML input to text

Flow Control

Flow control nodes control the flow of an XML pipeline. For example, you might choose to use a Stop node to display a message when the XML pipeline encounters an error condition – such as when it requires an XML document fails validation against its XML Schema.

You can use the following nodes to control the flow of an XML pipeline in Stylus Studio

- Choose – one or more IF conditions, and an ELSE condition
- Stop – stops XML pipeline processing, if, for example, generated XML does not validate against a given XML Schema
- Validate – uses XML Schema to validate XML
- Warning – displays a warning message in output, but allows XML pipeline processing to continue

Data Sources

Data source nodes are used to specify the XML data that is to be processed. For example, your XML pipeline might begin by processing raw XML, or it might require that non-XML data (such as a text file or a relational table) first be converted to XML prior to additional processing.

You can use the following nodes to specify data sources in an XML pipeline:

- ConvertToXML – specifies an operation that converts a flat file (CSV, binary, and so on) or EDI message type to XML.
- ConvertFromXML – specifies an operation that converts XML to some other format (CSV, binary, and so on).
- Pipeline Input – specifies an external input to an XML pipeline that includes the XML pipeline in which this node is defined.
- Pipeline Output – specifies an external output to an XML pipeline that includes the XML pipeline in which this node is defined.

Tip You can also provide a data source by specifying the **Default Value** property on that node's input port. For example, you could specify the URL of an XML document in this way.

Input and Output Ports

XML pipeline nodes are connected to each other by one or more pipes. The pipes represent the flow of XML data from one operation or transformation in the XML pipeline to another. Pipes connect to a node's input and output ports, which are found on most nodes representing XML pipeline components. (Not all nodes have both input and output ports.)

- You use the *input port* to specify the expected source for the node. You can specify a default value, or you can connect another node's output port to it with a pipe. For example, you might specify the input port for an XQuery node using the URL for an XML document, or as the output from a ConvertToXML node. If a pipe is connected to an input port, any default value is ignored.
- You use the *output port* to specify what to do with result from the node's processing. You can also specify output ports explicitly or implicitly. For example, you might specify the URL to which you want the output of a node be copied, you might link the output to a Validate node's input port, or you might do both.

You specify the flow of an XML pipeline's processing by linking one node's output port to another node's input port.

Identifying Resources

When planning your XML pipeline, you should consider the resources you will require in order to build it. These include

- Source documents for any XQuery and XSLT transformations the XML pipeline will use, XML Schema used for validation, and so on
- Data sources, whether an XML document, data converted from a relational database or flat file to XML, or XML data returned by a Web service, for example

You should pay particular attention to where these resources will be relative to the finished XML pipeline when it is being run in its production environment. For example, if you are using a Web service or relational database that requires some type of authentication, you need to ensure that the finished application that makes use of your XML pipeline code includes some means of providing authentication. Similarly, you need to ensure that the paths of source documents, data sources, and output URLs will be accessible to the finished application, or that your application provides a way to enter this information at run-time.

Deployment Considerations

When you are finished building and testing your XML pipeline, you will want to generate the code that executes the XML pipeline so that you can incorporate it in an application that uses the XML processing it defines.

In order to do this, you need to understand the environment in which the XML pipeline's application will be run and model that environment in Stylus Studio. For example, your XML pipeline might be required to use a given XML Schema validation engine, or a specific XQuery or FO processor, so you should plan for these requirements when designing your XML pipeline.

You choose the processors you want the components in your XML pipeline to use by specifying the **Execution Framework** settings on the **Deployment** page of the **Scenario Properties** dialog box.

Tip You can create multiple scenarios for the same XML pipeline and specify different execution frameworks for each.

See [“Specifying an Execution Framework”](#) on page 870 for more information.

Use Case: Building order.pipeline

This section describes the steps you might use to build the XML pipeline `order.pipeline`. This XML pipeline is in the `pipelines\order` folder in the `examples` project installed with Stylus Studio.

This section covers the following topics:

- [“order.pipeline Requirements”](#) on page 868
- [“Getting Started: Creating a New XML Pipeline”](#) on page 869
- [“XML Pipeline Scenarios”](#) on page 870
- [“Specifying an Execution Framework”](#) on page 870
- [“Configuring Data Sources”](#) on page 871
- [“Using XQuery to Merge Source File Data”](#) on page 876
- [“Adding an XQuery Node”](#) on page 881
- [“Setting the XQuery Node Data Sources”](#) on page 882
- [“Testing the XML Pipeline”](#) on page 883
- [“Setting a Value for an Output Port”](#) on page 883
- [“Designing a Report from the XML Document”](#) on page 885

- [“Adding XSLT and XQuery Transformations”](#) on page 887
- [“Finishing Up”](#) on page 892

order.pipeline Requirements

An organization in our enterprise has requested a report listing book orders, like the one shown in [Figure 356](#). The report must be available in both HTML and PDF formats. The source for the book order data is an EDIFACT message; inventory information is contained in a text file.

Book order:		
ISBN	Title	Quantity
0764569104	<i>XPath small TM/small 2.0 Programmer's Reference (Programmer to Programmer)</i>	25
0764569090	<i>XSLT 2.0 Programmer's Reference (Programmer to Programmer)</i>	25
1861004656	<i>Professional Java Server Programming J2EE Edition</i>	16
0596006756	<i>Enterprise Server Bus</i>	10

Figure 356. Sample Report Output Required for order.pipeline

The report consists of a table that lists the ISBN, title, and quantity of books that have been ordered from inventory.

Getting Started: Creating a New XML Pipeline

This section describes how to create a new XML pipeline document and some of the default behaviors for new XML pipelines.

- ◆ **To create an XML pipeline, select File > New > XML Pipeline from the Stylus Studio menu.**

When you create an XML pipeline, Stylus Studio displays a new .pipeline document in the XML Pipeline Editor. The document has a name of `untitledn.pipeline`, where *n* is a unique number.

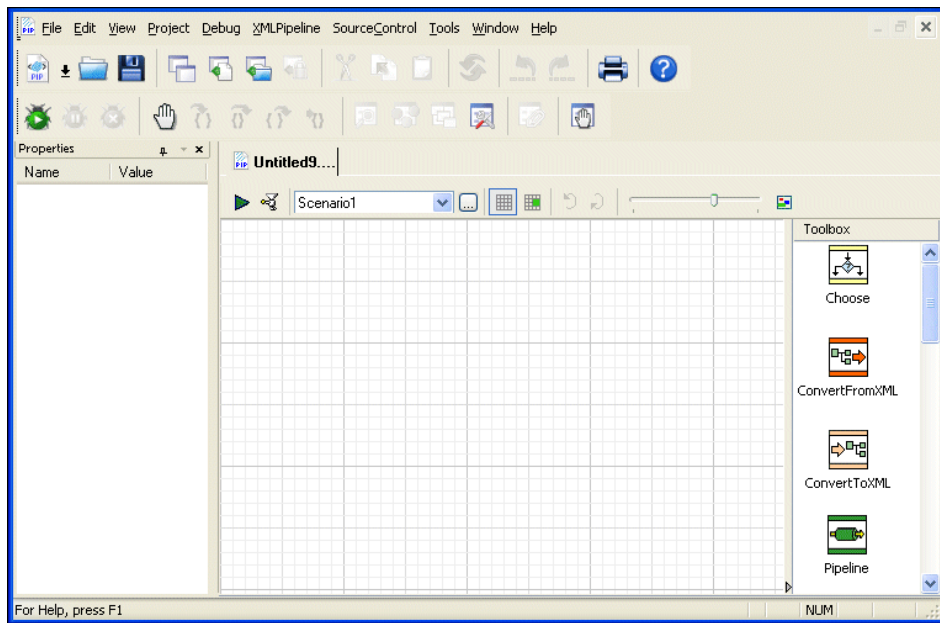



Figure 357. New XML Pipeline Document

Save the XML Pipeline

Click the Save button () and save the new XML pipeline (as `myOrders.pipeline`, for example).

XML Pipeline Scenarios

The XML pipeline document is associated with a default *scenario*, Scenario1. A scenario contains default deployment and processor settings that are used when

- Executing the XML pipeline in Stylus Studio
- Debugging the XML pipeline in Stylus Studio
- Generating code for the XML pipeline

You can define multiple scenarios using different settings to see how each affects XML pipeline processing.

Specifying an Execution Framework

To help you manage processor settings for the XQuery, XSLT, XML Schema validation, and FO processing operations in your XML pipeline, Stylus Studio provides an *execution framework* on the **Deployment** tab of the **Scenario Properties** dialog box.

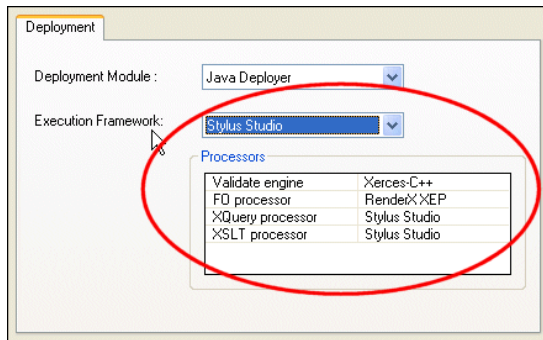


Figure 358. Execution Framework Settings for an XML Pipeline

Each execution framework is associated with a pre-set collection of compatible processors. You can

- Change the execution framework to use a different set of processors.
- Change the settings for individual processors within an execution framework. Any changes you make to settings within an execution framework affect the current pipeline only, and not the execution framework's default settings.

When to Specify the Execution Framework

If the production or deployment environment anticipated for the XML pipeline uses different processors than those specified in the default execution framework, you should consider changing these settings early in the XML pipeline's development phase. Doing so will enable you to preview and debug the XML pipeline's performance and output in an environment that models the production environment as closely as possible. In any event, you need to make sure that the **Processor** settings on the **Deployment** page of the **Scenario Properties** dialog box are set appropriately prior to generating code for your XML pipeline. See [“Generating Code for an XML Pipeline”](#) on page 920 for more information on this topic.

Configuring Data Sources

The source for the information required for the desired report comes from two files:

- A flat file, `booksXML.txt`, that contains ISBN, title, manufacturer, and release date information for every book in inventory.
- An EDI file, `order.edi`, that contains title, ISBN, and quantity information for every book currently on order.

Neither of these files provides data in XML format, so they will have to be converted to XML. We will use built-in DataDirect XML Converters for this task.

Ways to Configure Non-XML Data Sources

You use ConvertToXML nodes to specify a non-XML data source in an XML pipeline. There are two ways to do this:

- Convert the source file using a built-in DataDirect XML Converter or a user-defined custom XML conversion, and then drag the resulting document and drop it on the XML pipeline canvas.
- Create the ConvertToXML node in the XML pipeline using the **Toolbox**, and then specify the source file and the DataDirect XML Converter or user-defined custom XML conversion you want to use to convert it to XML.

While both require a similar number of steps, converting a source file can be more economical as it creates a resource that you can reuse in the XML pipeline, and elsewhere in Stylus Studio. Both procedures are described in the following sections.

Convert booksXML.txt Using a Built-in XML Converter

◆ **To convert booksXML.txt using a built-in XML Converter:**

1. Select **File > Open** from the menu.
The **Open** dialog box appears.
2. Navigate to the `examples\pipeline\order` folder where you installed Stylus Studio.
3. Change the **Files of type** field to **All Files**.
4. Select `booksXML.txt`.
5. Select the **Open using XML Converter** check box at the bottom of the Open dialog box.
6. Click **Open**.
Stylus Studio displays the **Select XML Converter** dialog box.
7. Select the **Comma-Separated Values** converter.
The `booksXML.txt` source file happens to use a vertical bar (|) as its separator character.

Tip

To view the source `.txt` and `.edi` files, double-click them in the **Project** window to display them in a Stylus Studio editor.

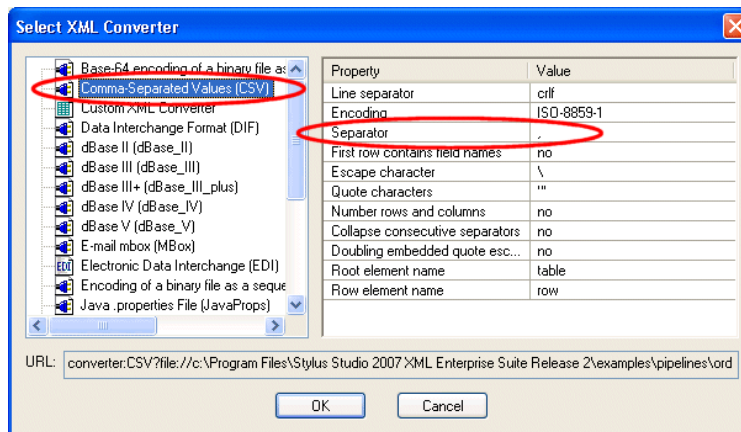


Figure 359. Select XML Converter Dialog Box

8. Change the value in the **Separator** property to a vertical bar (|).
9. Change the value in the **First row contains field names** to Yes.

10. Click **OK**.

The booksXML.txt file appears in the **Other Documents** folder in the Project window. If you hover the mouse pointer over the file name, you will see the full specification of the XML Converter URL used to convert it.

11. Drag booksXML.txt and drop it on the XML pipeline canvas.

Stylus Studio creates a ConvertToXML node with its input port already specified.

12. Go to [“Create a ConvertToXML Node for order.edi”](#) on page 875.

Create a ConvertToXML Node for booksXML.txt

◆ To create XML data from a non-XML Source in the XML Pipeline Editor:


1. Drag the ConvertToXML icon from the **Toolbox** pane and drop it on the XML pipeline canvas.

Stylus Studio creates a ConvertToXML node and displays it in the XML pipeline diagram.

2. Display the **Properties** window (**View > Properties**) if it is not already open.3. Display the **Project** window (**View > Project**) if it is not already open.

4. Click the input port on the ConvertToXML node.

The properties for the input port are displayed in the **Properties** window.

5. Click the **Default Value** field; click the more button () when it appears.

Stylus Studio displays the **Default Value** dialog box.

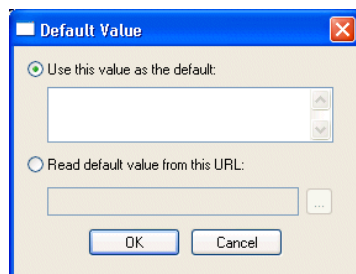


Figure 360. Default Value Dialog Box

6. Click the **Read default value from this URL** radio button, and then navigate to examples\pipelines\order\ folder where you installed Stylus Studio and select booksXML.txt.

7. Click **OK**.

The input port on the ConvertToXML Node turns green, indicating that a source document has been specified as the input.

Next, we need to specify which built-in converter to use to convert booksXML.txt to XML.

8. Click the ConvertToXML node in the XML pipeline diagram.

9. Click the **XML Converter URL** field in the **Properties** window; click the more button (...)

Stylus Studio displays the **Select XML Converter** dialog box.

10. Select the Comma-Separated Values converter.

The booksXML.txt source file happens to use a vertical bar (|) as its separator character.

Tip

To view the source .txt and .edi files, double-click them in the **Project** window to display them in a Stylus Studio editor.

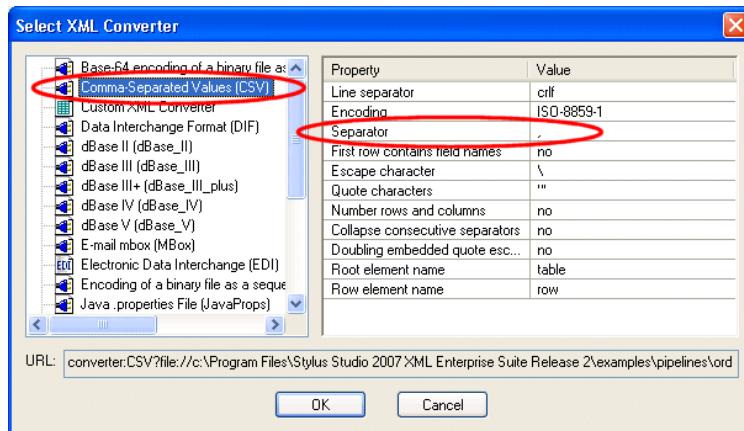


Figure 361. Select XML Converter Dialog Box

11. Change the value in the **Separator** property to a vertical bar (|).

12. Change the value in the **First row contains field names** to Yes.

13. Click **OK**.

The ConvertToXML node representing the conversion of the booksXML.txt file to XML is now completely specified, except for the output. We will address that in the section [“Setting the XQuery Node Data Sources”](#) on page 882.

Create a ConvertToXML Node for order.edi

Next, using the procedure described in either “[Convert booksXML.txt Using a Built-in XML Converter](#)” on page 872 or “[Create a ConvertToXML Node for booksXML.txt](#)” on page 873, create a ConvertToXML node for the order.edi file. Note the following changes:

- Use select order.edi as the source document.
- Select **Electronic Data Interchange (EDI)** from the **Select XML Converter** dialog box. Accept all default values.

Renaming Nodes

When you add a second node of the same type using the **Toolbox**, Stylus Studio gives the second node the same name as the first, with a number to make it unique. If you create a node by dragging a document and dropping it on the canvas, Stylus Studio gives the node the same name as the file.

Depending on how you created the ConvertToXML nodes in the XML pipeline, they are named either:

- *Convert to XML* and *Convert to XML #2*, or
- *booksXML.txt* and *order.edi*

You can rename nodes using the following procedure:

◆ To rename a node:

1. Select the node you want to rename in the XML pipeline diagram.
2. Click the value in the **Name** field in the **Properties** window.
3. Type the new name and press Enter.

Tip Node names are displayed as tooltips in the XML pipeline diagram when you hover the pointer over the node. You can also create labels for the diagram and for nodes within the diagram. See “[Labeling](#)” on page 911 for more information.

The XML Pipeline So Far

At this point, we have defined the converters to be used to convert our non-XML data sources to XML. The resulting XML pipeline, `myOrders.pipeline`, looks something like this:

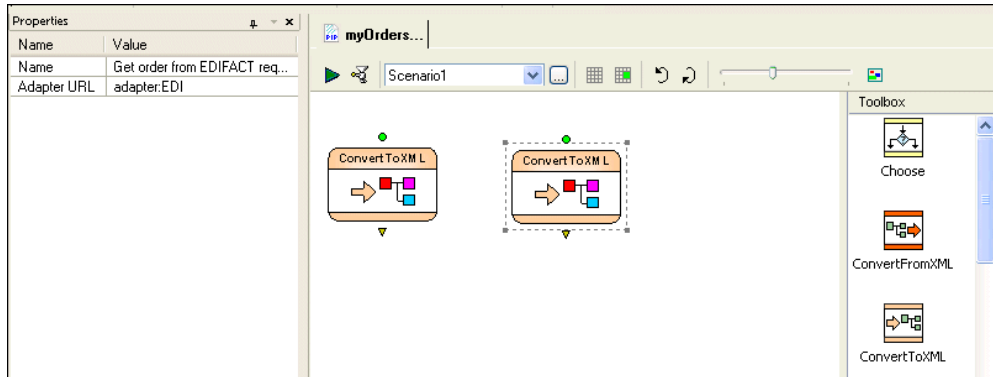


Figure 362. myOrders.pipeline After Defining Non-XML Data Sources

In the next stage of the process, we will specify the XQuery file that we will use to join the data from these two files.

Using XQuery to Merge Source File Data

As described in “[order.pipeline Requirements](#)” on page 868, the data required for our report comes from two files – a text file, `booksXML.txt`, and an EDI file, `order.edi`. Both files have the book’s ISBN, and we will use that number to select matching data. This will provide us with the following information required by our report:

- Book title (from `booksXML.txt`)
- ISBN and quantity (from `order.edi`)

To join data from these different documents we will use an XQuery document created using the XQuery Mapper. Once we have defined that XQuery document, we can add the XQuery node to our XML pipeline.

Using Variables to Reference Data Sources

Because we want our XQuery to be easy to parameterize, we will create it using external variables to reference our two data sources, `booksXML.txt` and `order.edi`. Once the data sources for the XQuery are defined, we can use the XQuery Mapper to map desired nodes

from these source documents to nodes in the XML Schema that represents the structure to which the resulting XML must conform.

If you open `createFullOrder.xquery`, which is in the `pipeline\order` folder of the examples project where you installed Stylus Studio, you can see how this was done. As shown in [Figure 363](#), `createFullOrder.xquery` uses variable declarations, `$ediOrder` and `$allBooks`, for the non-XML data sources. DataDirect XML Converters, which are built in to Stylus Studio 2007 XML Enterprise Suite, convert these data sources to XML on-the-fly, any time the XQuery (or an XML pipeline that uses the XQuery) is executed. Thus, if the data in either `booksXML.txt` and `order.edi` changes, the report resulting from the XML pipeline will change, too.

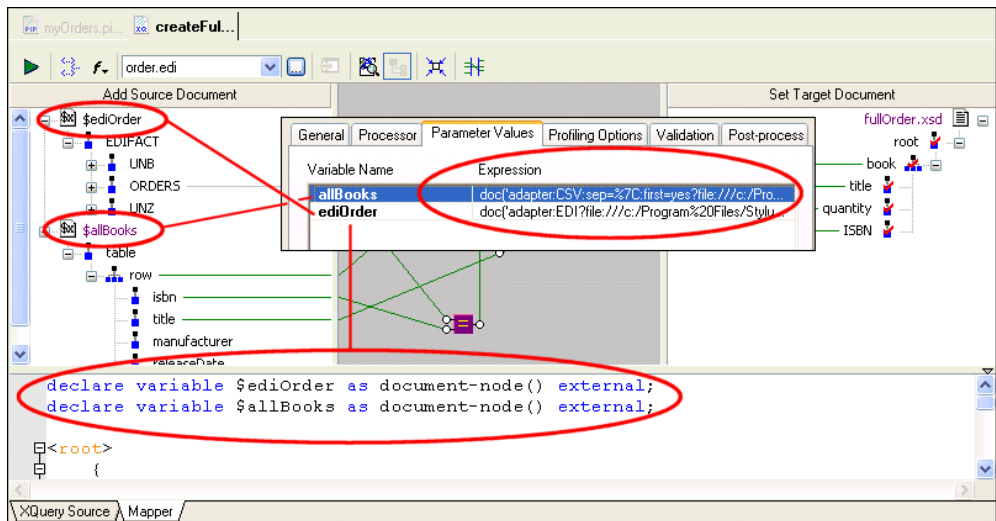


Figure 363. Variables Created for Non-XML Data Sources

◆ **To use non-XML as a source document for XQuery Mapper:**

To specify a non-XML data source as an XML source for the XQuery Mapper, you

1. Click the **Add Source Document** button at the top of the XQuery Mapper tab. This displays the **Open** dialog box.

2. You select the file you want to open, and then select the **Open using XML Converter** check box.

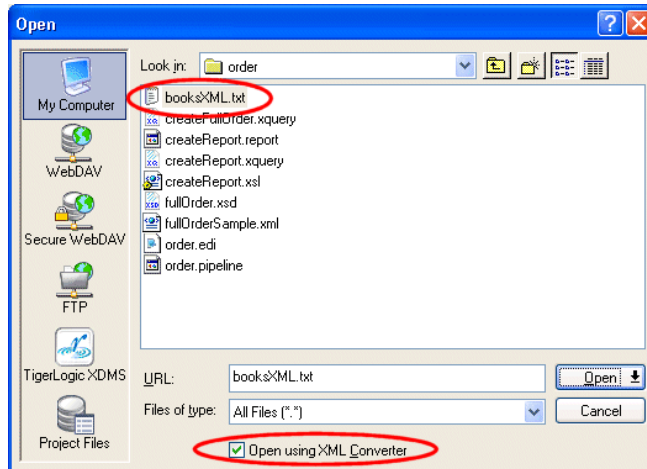


Figure 364. Using Non-XML Data as an XQuery Data Source

3. When you click **Open**, Stylus Studio displays the **Select XML Converter** dialog box (see [Figure 361](#)), which you use to select the built-in DataDirect XML Converter you want to use to convert your non-XML file to XML. Note that this is the same procedure we used to identify the data sources for the XML pipeline's two ConvertToXML nodes.
4. When you click OK on the **Select XML Converter** dialog box, Stylus Studio adds it as a data source in the **Add Source Document** pane of the XQuery Mapper.

Next, we need to create global variables for the two source documents. This way, the XQuery code that Stylus Studio generates will use variable declarations instead of document functions to reference our data sources.

◆ **To associate the source schema with a global variable:**

1. Right click the source document name and select **Associate With > Global Variable**. Stylus Studio displays the **Associate Schema with Variable** dialog box.

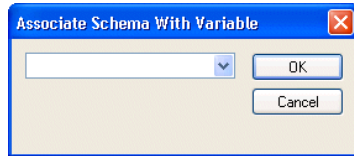


Figure 365. Associate Schema with Variable Dialog Box

2. Enter the value you want to use for the variable, and click OK. When a variable is created (allBooks, for example), Stylus Studio creates a declaration like the following in the XQuery source code:

```
declare variable $allBooks as document-node() external;
```

Looking at the XQuery Code

Before moving on with the XML pipeline creation, let's a quick look at the XQuery code in createFullOrder.xquery:

```
declare variable $ediOrder as document-node() external;
declare variable $allBooks as document-node() external;

<root>
{
  for $GROUP_28 in $ediOrder/EDIFACT/ORDERS/GROUP_28,
   $row in $allBooks/table/row
  where $GROUP_28/LIN/LIN03/LIN0301/text() = $row/isbn/text()
  return
  <book>
    <title>
      {$row/title/text()}
    </title>
    <quantity>
      {$GROUP_28/QTY/QTY01/QTY0102/text()}
    </quantity>
    <ISBN>
      {$GROUP_28/LIN/LIN03/LIN0301/text()}
    </ISBN>
  </book>
}
</root>
```

The first two lines contain the variable declarations for `booksXML.txt` and `order.edi`, our two source files. A FLWOR block (For, Let, Where, Order by, Return) matches ISBN numbers from `order.edi` with those in `books.xml`, and when it finds a match, it returns

- The title (from `booksXML.txt`)
- The quantity (from `order.edi`)
- The ISBN (from `order.edi`)

All of this code was created automatically as a result of mapping nodes from our source documents to nodes in an XML Schema, `fullOrder.xsd`, which was provided by the organization in our enterprise that requested the inventory report – all XML resulting from the `createFullOrder.xquery` must conform to this XML Schema.

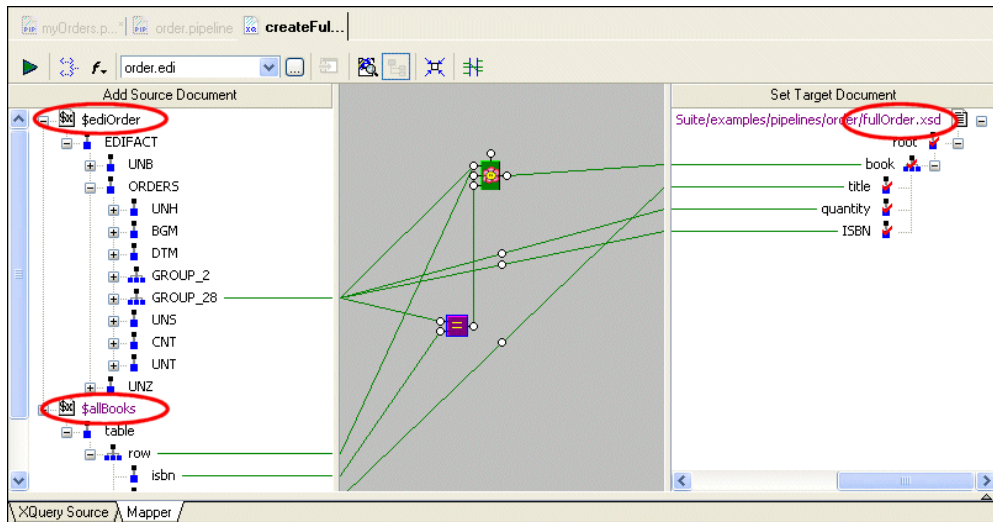


Figure 366. XQuery Mapper Used to Generate `createFullOrder.xquery`

See [“Building an XQuery Using the Mapper”](#) on page 750 for more information on using the XQuery Mapper.

Adding an XQuery Node

Now that we understand how the XQuery code in `createFullOrder.xquery` uses a FLWOR expression to join data from our data sources, we can add it to our XML pipeline.

◆ **To add an XQuery node to an XML pipeline:**

1. Drag the XQuery icon from the **Toolbox** pane and drop it on the XML pipeline canvas. Stylus Studio creates an XQuery node and displays it in the XML pipeline diagram.
2. Display the **Properties** window (**View > Properties**) if it is not already open.
3. Display the **Project** window (**View > Project**) if it is not already open.
4. Drag `createFullOrder.xquery` and drop it on either
 - The XQuery node on the XML pipeline diagram
 - The **Value** field for the **.xquery file** property

The XQuery node now has two additional input ports, one named `allBooks`, and the other named `ediOrder`.

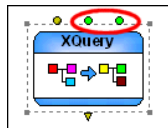


Figure 367. XQuery Node for createFullOrder.xquery

These input ports are colored green, indicating that each has a default value specified for it. These default values correspond to the two data sources we specified as source documents in the XQuery Mapper.

Tip

If you drop `createFullOrder.xquery` directly on the XML pipeline canvas, Stylus Studio automatically creates the `ConvertToXML` nodes that represent the data sources it uses. See [“An Alternate Way to Create ConvertToXML Nodes”](#) on page 883.

5. Change the XQuery node’s default name, XQuery operator, to `Extract full order information`. (See [“Renaming Nodes”](#) on page 875 if you need help with this step.)

Changes to Source Documents

XML pipelines reference external documents, like the `createFullOrder.xquery` document we just added. They do not create copies of these documents. Therefore, when changes to a source document are saved, the XML pipeline picks up these changes the next time it is executed.

Setting the XQuery Node Data Sources

Although the XQuery code was specified with default data sources, we want the XQuery to use the data sources we defined in the two `ConvertToXML` nodes we created in “[Configuring Data Sources](#)” on page 871. We do this by connecting the output ports on the `ConvertToXML` nodes to the input ports on the XQuery node.

◆ **To set an XQuery node’s data sources:**

1. With the pointer over the `Get books catalog from txt file` `ConvertToXML` node, drag and drop the output port on the `allBooks` input port on the `Extract full order information` XQuery node.

Stylus Studio creates a pipe connecting the two nodes, shown in [Figure 368](#).

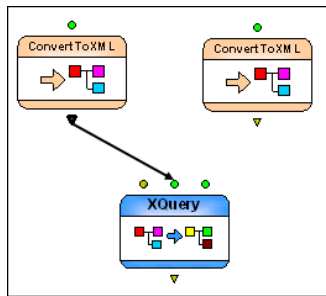


Figure 368. The XML Pipeline’s First Pipe

2. Repeat this procedure, connecting the output port from the `Get order from EDIFACT request` `ConvertToXML` node to the `ediOrder` input port on the `Extract full order information` XQuery node.

Default and Specified Port Values

As you can see in [Figure 368](#), an input port can have both a default value, and a value provided by another node's output port. Note, however, that an output port's default value is never used if a pipe is connected to the port – the pipe either supplies a value or it does not, but the default value is ignored.

An Alternate Way to Create ConvertToXML Nodes

It should be noted that when you add an XQuery or an XSLT document to an XML pipeline, Stylus Studio creates other nodes required to support the resulting XQuery or XSLT node, based on the XQuery or XSLT definition. For example, since `createFullOrder.xquery` was defined to use Stylus Studio's built-in converters to create XML data source documents, simply dragging and dropping the `createFullOrder.xquery` on the XML pipeline canvas would have automatically created the XQuery node and both ConvertToXML nodes, one for each data source specified in the XQuery code.

We will see this functionality in action later, when we add another XQuery document to the XML pipeline. (See [“Add createReport.xquery”](#) on page 889 for more information.)

Testing the XML Pipeline

The XML pipeline as it is defined now creates an XML document containing a book parent node, with `title`, `quantity`, and `ISBN` child nodes. Let's test it before continuing.

◆ **To test an XML pipeline, click the Execute button (▶):**

In this case, Stylus Studio displays a message indicating that the XML pipeline, as it is currently defined, does not have an output. And if we examine our XML pipeline, we see that this is true – processing terminates with the XQuery node, but its output port is empty.

Setting a Value for an Output Port

To quickly verify that our XML pipeline works as expected, we can create an output for the XQuery node's output port.

◆ **To set a value for an output port:**

1. Click the output port.
2. Specify a value for the **Copy To URL** property (`myFullOrderSample.xml`, for example).
The output port changes color, indicating that it has a value specified for it.

If we test the XML pipeline again, we can see that the XML pipeline runs to completion. Stylus Studio displays the **Preview** window, the **Main** tab of which displays an execution log that describes the processing steps executed in the XML pipeline.

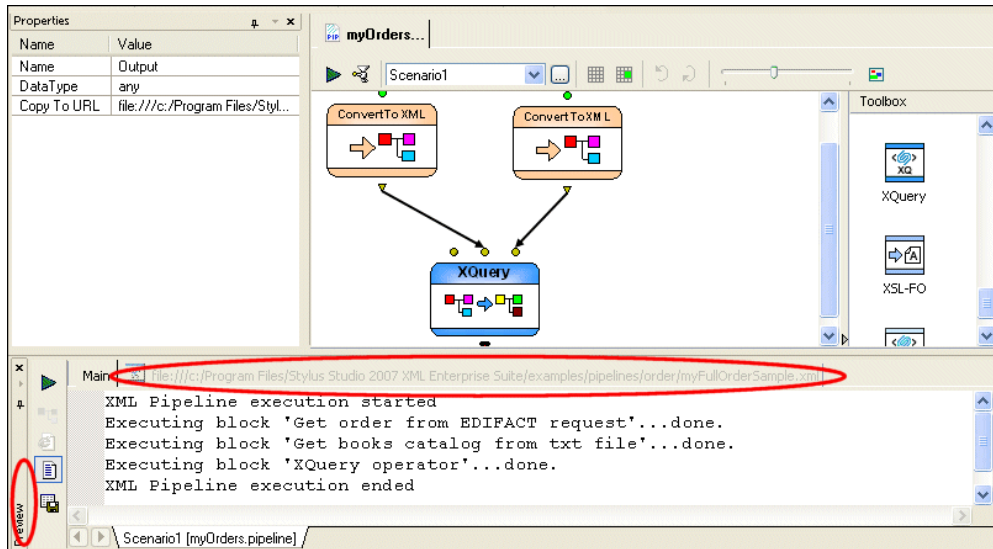



Figure 369. XML Pipeline Execution Messages in Preview Window

Tip If you click an entry in the execution log, Stylus Studio's back-mapping feature highlights node responsible for that processing step.

If we click the next tab in the **Preview** window, we can see the XML output by our XML pipeline in text view. If we click the **Preview in Tree** button (), we can verify that the XML document is of the structure we expect.

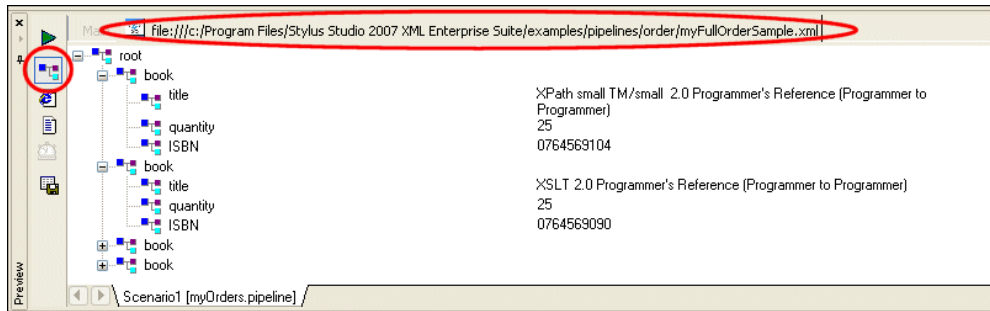


Figure 370. Tree View of XML Pipeline Output

Tip You could also click the XQuery node's output port to display this tab.

Designing a Report from the XML Document

Now that we have an XML document that represents our joined data sources, we need to develop finished reports in HTML and PDF. The Stylus Studio *XML Publisher* helps you design reports based on XML documents or XML Schema, and then generate XQuery or XSLT code to create that report in HTML+CSS or XSL-FO document formats.

As shown in [Figure 371](#), the XML Publisher createReport.report uses the XML document resulting from our XML pipeline, myFu10orderSample.xml, as the data source to design a simple book order report.

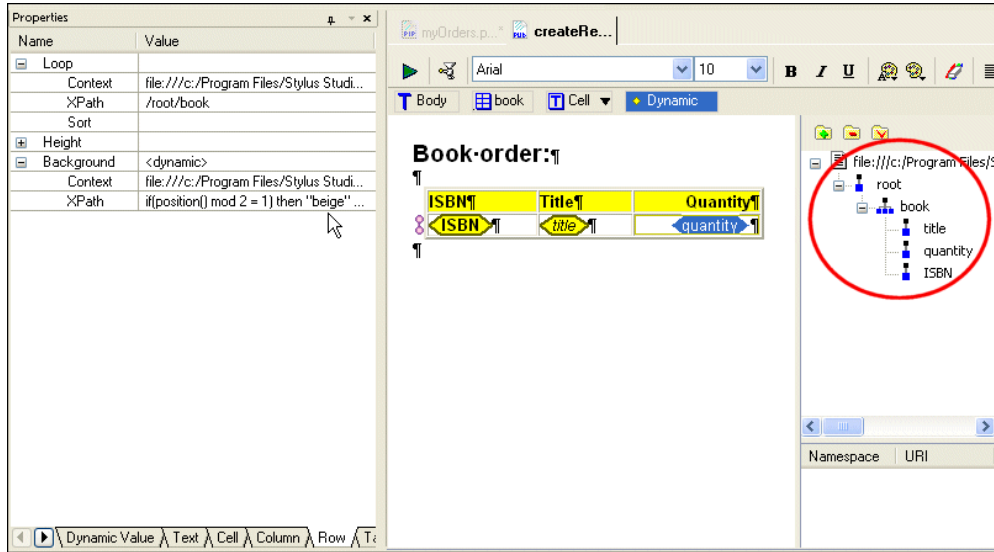


Figure 371. XML Publisher Report Designer

The table, and the values in its columns, was created by simply dragging nodes from the source document tree and dropping them on the XML Publisher canvas. Additional formatting was specified using XPath expressions (to control row color and quantity color, for example).

When the report design was complete, we used XML Publisher to generate XQuery and XSLT code using the **Generate Transformation** dialog box.

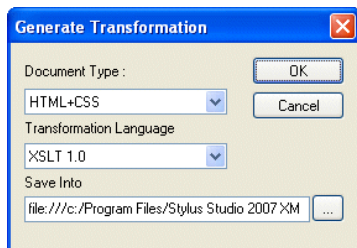


Figure 372. XML Publisher Generate Transformation Dialog Box

We created `createReport.xquery` in one generate pass, and `createReport.xsl` in another. We will add these transformations to our XML pipeline next.

See [Chapter 13, “Publishing XML Data,”](#) to learn more about designing reports using the XML Publisher.

Adding XSLT and XQuery Transformations

The next step in building our XML pipeline is to add the XSLT and XQuery transformations generated using XML Publisher. These transformations will render the XML document resulting from the `Extract full order information XQuery`, `myFullOrderSample.xml`, as HTML and PDF, respectively.

Of course, if we wanted to, we could have used XQuery to generate the HTML and XSLT to generate the XSL-FO; or we could have used just XQuery or XSLT to generate both document formats. The technology you choose is largely a matter of personal preference, though some are better suited to certain tasks (like data aggregation, HTML formatting, and so on) than others.

Add `createReport.xsl`

◆ **To add `createReport.xsl` to the XML pipeline:**

1. Display the **Project** window (**View > Project**) if it is not already open.
2. Drag `createReport.xsl` from the `pipelines/order` folder and drop it on the XML pipeline canvas.

Stylus Studio add an XSLT node to the XML pipeline.

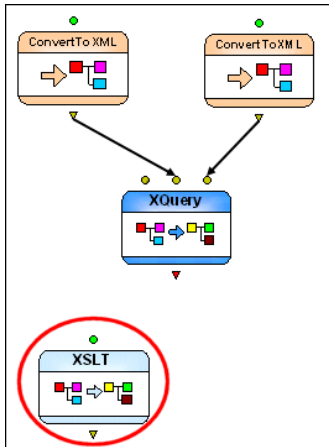


Figure 373. New XSLT Node

The colored input port indicates that this XSLT node already has a default input and value defined for it. We will specify our own input value (the XML document created by the Extract full order information XQuery node).

Tip Stylus Studio uses the file name for the node name when you create the node by dragging and dropping a document on the XML pipeline canvas.

3. Drag a pipe from the output port on the Extract full order information XQuery node to the input port on the new XSLT node.
4. Next, specify a value for the output port (order.html, for example). See [“Setting a Value for an Output Port”](#) on page 883 if you need help with this step.

Tip Since we know the XQuery node generates the XML document we require, we can delete the **Copy to URL** for its output port. Otherwise, output will continue to be written to that URL.

5. Test the XML pipeline by clicking the **Execute** button (▶).

As currently defined, our XML pipeline should create an HTML report based on the `myFullOrderSample.xml` document, and this is what appears in the **Preview** window.

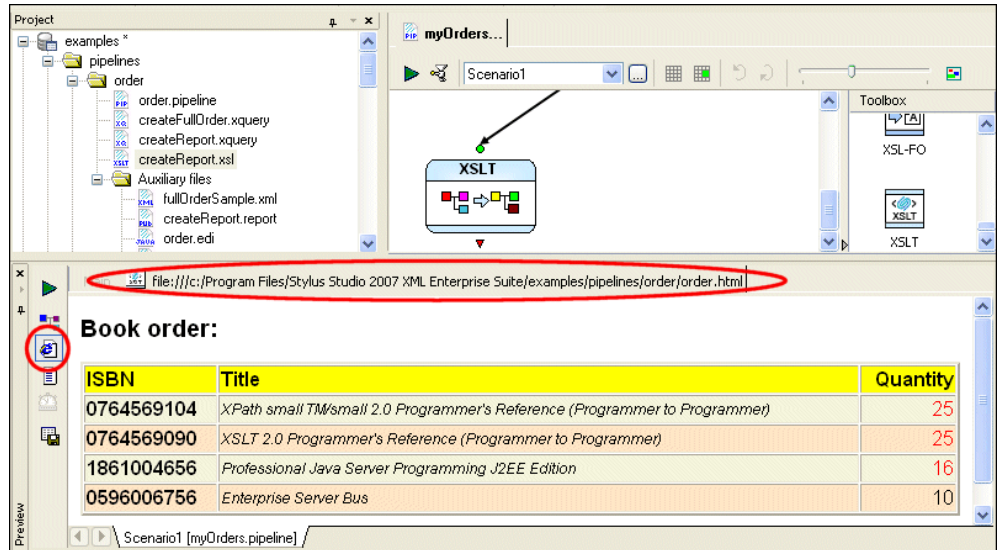


Figure 374. Preview of the XML Pipeline's HTML Output

Tip Click the **Preview in Browser** button to see the XML pipeline output rendered in HTML.

Add createReport.xquery

All that remains for our XML pipeline definition is to specify the XQuery node that will transform `myFullOrderSample.xml` into PDF. For this purpose, we will use the `createReport.xquery` generated by the XML Publisher.

◆ To add `createReport.xquery` to the XML pipeline:

1. Display the **Project** window (**View > Project**) if it is not already open.
2. Drag `createReport.xquery` from the `pipelines/order` folder and drop it on the XML pipeline canvas.

Stylus Studio may display a warning message, indicating that the processor specified for the `createReport.xquery` document differs from that specified for the XML pipeline.

3. Click **OK** to accept the default recommendation. (See [“Managing Processor Conflicts”](#) on page 897 for more information on this topic.)

Stylus Studio adds a new XQuery and associated XSL-FO node to the XML pipeline.

The XSL-FO node is the result of the post-processing specified for the XQuery – when we generated the XQuery code from XML Publisher, we chose XSL-FO for the **Document Type** (see [Figure 372](#)). Stylus Studio automatically selected the default FO processor, RenderX XEP, to process the FO generated by `createReport.xquery`.

4. Drag a pipe from the output port on the Extract full order information XQuery node to the input port on the new XQuery node.

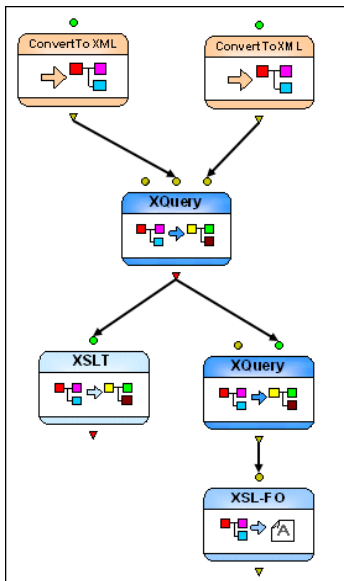


Figure 375. New XQuery Node with Automatically Created XSL-FO Node

5. Next, specify a value for the output port (`order.pdf`, for example) of the XSL-FO node. See [“Setting a Value for an Output Port”](#) on page 883 if you need help with this step.

6. Test the XML pipeline one last time by clicking the **Execute** button (▶). Stylus Studio reopens the **Preview**, which displays an execution log for the XML pipeline's operations. New statements appear for the new XQuery and XSL-FO nodes.

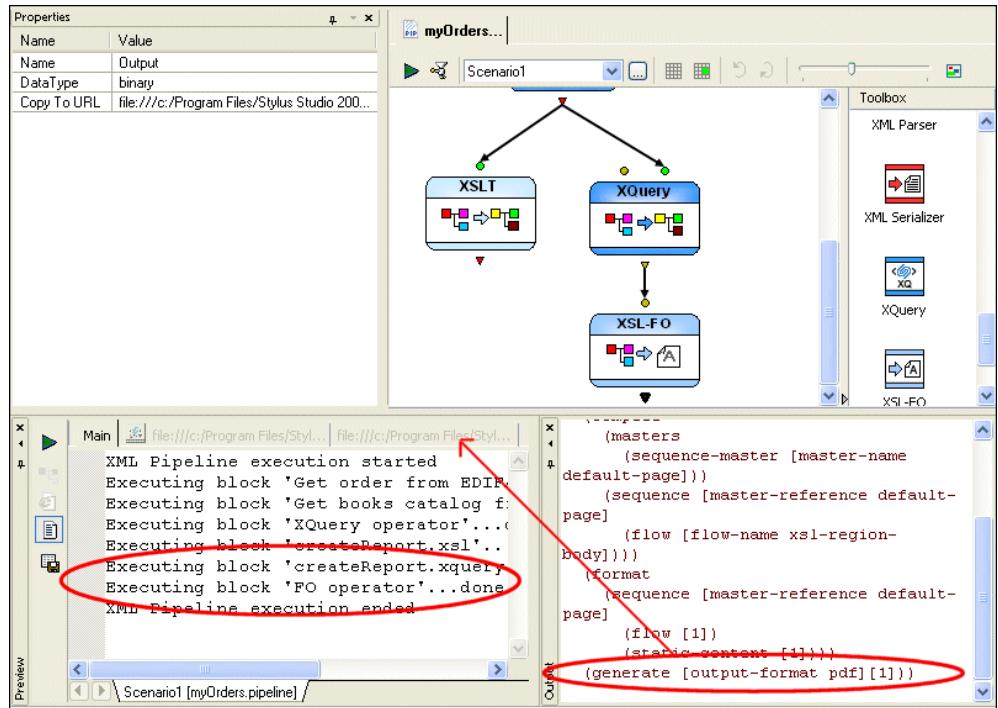


Figure 376. Pipeline Execution Log and FO Processor Output

It also reopens the **Output** window, which shows output from the XSLT processor and the RenderX FO post-processor.

- Click the second tab in the **Preview** window to display the PDF document created by the nodes we just added to the XML pipeline.

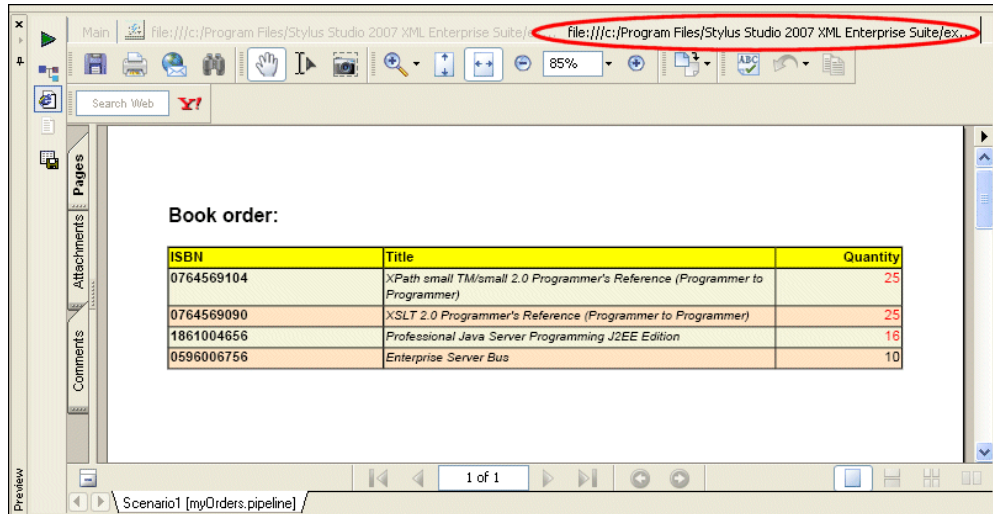


Figure 377. PDF Created by the XQuery and XSL-FO Nodes

Note If you receive an error during this step, it might mean that Adobe Acrobat Reader was not properly installed on your system. Go to <http://www.adobe.com> and reinstall Acrobat Reader.

Finishing Up

Now that the XML pipeline has been tested and we have seen that it successfully generates the order report in both HTML and PDF document formats, we can generate Java code for the entire XML pipeline. This code could then be incorporated in, say, a Java application.

See “[Generating Code for an XML Pipeline](#)” on page 920 for more information on this topic.

Working with Nodes

This section describes how to work with the different nodes you might use in an XML pipeline. It references XML pipelines in the pipelines folder in the examples project.

This section covers the following topics:

- [“Types of Nodes”](#) on page 893
- [“Adding Nodes to an XML Pipeline”](#) on page 893
- [“XQuery and XSLT Nodes”](#) on page 896
- [“XSL-FO Nodes”](#) on page 898
- [“ConvertToXML and ConvertFromXML Nodes”](#) on page 905
- [“Validate Nodes”](#) on page 901
- [“Stop and Warning Nodes”](#) on page 907
- [“Pipeline and Related Nodes”](#) on page 899
- [“Choose Nodes”](#) on page 903
- [“XML Parser Nodes”](#) on page 909
- [“XML Serializer Nodes”](#) on page 910

For information about node properties, see [“XML Pipeline Node Properties Reference”](#) on page 924.

Types of Nodes

Nodes can be grouped into three broad categories:

- Transformation
- Flow control
- Data sources

These categories are described in [“XML Pipeline Components”](#) on page 864.

Adding Nodes to an XML Pipeline

There are two ways to add nodes to an XML pipeline. You can

- Use existing documents
- Use the icons in the **Toolbox** pane

Using Existing Documents

To create a node in an XML pipeline using an existing document, just drag and drop the document (from the **Project** window or the **File Explorer**, for example) on the XML pipeline canvas. This creates a node that represents the document you dropped on the canvas. For example, if you drag and drop an XQuery document, Stylus Studio creates an XQuery node based on that XQuery document. It also incorporates that XQuery document's default scenario properties settings, including its input and output URLs, processor, and post-process instructions, validation instructions, and so on.

In addition to dropping a file on the XML pipeline canvas, you can also drop it on an existing node or on the appropriate **Value** field for that node displayed in the **Properties** window. A node's input and output ports cannot be used as drop targets.

Using the Toolbox

To create a node in an XML pipeline using the **Toolbox**, just drag an icon from the **Toolbox** pane and drop it on the XML pipeline canvas.

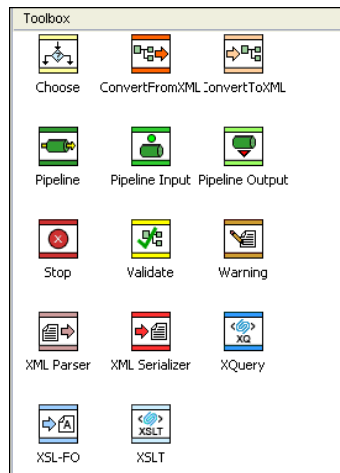


Figure 378. XML Pipeline Toolbox

This creates a node that is not yet implemented, whose properties you then need to specify – either by typing, or by dragging an external document and dropping it on the node or one of its **Value** fields in the **Properties** window. For example, if you drag the XSLT icon and drop it on the canvas, you would need to specify values for the input and output nodes, as well as the URL of the XSLT document you want that node to represent in your XML pipeline.

Available tools are described in the following table.

Table 103. XML Pipeline Tools

<i>Tool</i>	<i>Description</i>
Choose	Used to direct the flow of XML pipeline processing using one or more XPath expressions.
ConvertFromXML	Used to convert XML to some other format (CSV, binary, and so on).
ConvertToXML	Used to convert a flat file (CSV, binary, and so on) or EDI message type to XML.
Pipeline	Represents an XML pipeline file you want to include in the another pipeline.
Pipeline Input	Used to specify an external input to an XML pipeline that includes the XML pipeline in which this node is defined.
Pipeline Output	Used to specify an external output to an XML pipeline that includes the XML pipeline in which this node is defined.
Stop	Used to terminate XML pipeline processing.
Validate	Used to validate XML specified as input using one or more XML Schemas.
Warning	Used to display a warning message during XML pipeline processing.
XML Parser	Used to convert text input to XML.
XML Serializer	Used to convert XML input to text.
XQuery	Used to query XML input using an XQuery.
XSL-FO	Used to process XML input using XSL-FO.
XSLT	Used to transform XML input using XSLT.

Node and Port Names

The default node name is a variation of the name as it appears in the **Toolbox** pane – the default name for the XSL-FO operation is *FO Operator*, for example. If you use an

existing document to create a node, Stylus Studio uses the file name as the default node name.

Node names are used for documentation purposes only; they do not affect XML pipeline execution, though they do appear as strings in the generated code and in messages in the XML pipeline execution log displayed in the **Preview** window. Node names appear in the XML pipeline as tooltips when you place the pointer on the node.

Tip You can create a label for a node that is always visible in the XML pipeline diagram. See “[Labeling](#)” on page 911 for more information.

XQuery and XSLT Nodes

XQuery and XSLT nodes represent XQuery and XSLT documents. XQuery and XSLT code is executed using the processors specified in the XML pipeline’s execution framework. See “[Specifying an Execution Framework](#)” on page 870 for more information on this topic.

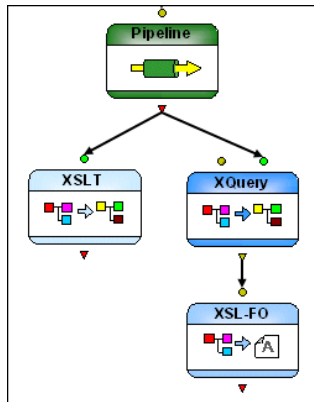


Figure 379. XQuery and XSLT Nodes in getHoldings.pipeline

Input Ports

XQuery and XSLT nodes have a single input port by default which you use to specify the XML input to be transformed. You can specify a default value, or the value can be dynamic (the output from another node in the XML pipeline, for example).

Additional input ports, like the one on the XQuery node in [Figure 379](#), appear if

- One or more external variables are defined for it (XQuery)
- One or more global parameters are defined for it (XSLT)

Output Ports

XQuery and XSLT nodes have a single output port, used to specify what to do with the result of the transformation. You can

- Use the **Copy to URL** property to write the output to a file system
- Pipe the output to one or more other nodes
- Or both

Scenario Properties

An XQuery or XSLT document's default scenario properties are reflected in the XML pipeline only if you drop the document directly on the canvas. For example, if the default scenario for your XQuery document specifies RenderX post-processing, Stylus Studio automatically adds an XSL-FO node to the XML pipeline, connected to the XQuery node's output port when you drop it on the canvas to represent the post-processing. If you drop the document on an existing XQuery node, however, the XSL-FO post-processing node is not added to the XML pipeline.

Scenario properties treated in this way include

- Input and output URLs
- Values for existing parameters and variables
- Processors
- Post-processing
- Validation

Similarly, changes made to default scenario properties are not reflected in the XML pipeline unless you re-add the document to the XML pipeline by dropping it on the canvas.

Changes to Source Code

When you save changes to the source XQuery or XSLT documents used in an XML pipeline, those changes are reflected the next time the XML pipeline is executed. If you added external variables or parameters, new input ports are added to the XQuery and XSLT nodes.

Managing Processor Conflicts

Each XQuery and XSLT document is associated with its own set of processors, which are specified on the **Processor**, **Post-Process**, and **Validation** tabs of XQuery and XSLT

Scenario Properties dialog boxes. When you add an XQuery or XSLT node to an XML pipeline by dragging and dropping the document onto the XML pipeline canvas, Stylus Studio displays the **Processor Mismatch** dialog box if the document's processor settings differ from those specified for the XML pipeline.

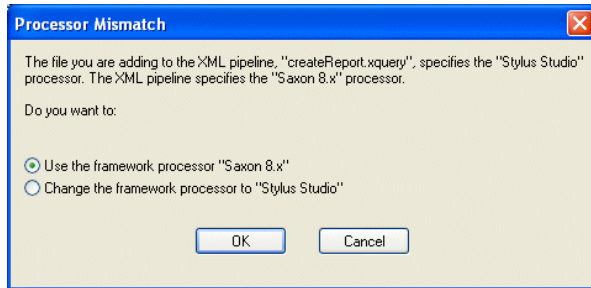


Figure 380. Processor Mismatch Dialog Box

You can

- Use the processor settings specified for the XML pipeline on the **Deployment** tab of the **Scenario Properties** dialog box
- Change the XQuery or XSLT processor setting for the XML pipeline

Note Stylus Studio checks XQuery and XSLT processor compatibility only. Validation and FO processing engines are not checked, and are not altered by any actions you take on the **Processor Mismatch** dialog box.

XSL-FO Nodes

XSL-FO nodes convert their input to PDF using the FO processor specified in the XML pipeline's execution framework. See [“Specifying an Execution Framework”](#) on page 870 for more information on this topic.

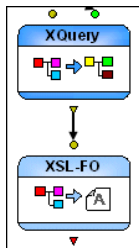


Figure 381. XSL-FO Node in getHoldings.pipeline

They are created automatically when an XQuery or XSLT document that specifies post-processing is dropped directly onto the XML pipeline canvas. You can also create one by dragging the XSL-FO icon from the **s**. The only property you can specify for an XSL-FO node is its name.

Input Port

XSL-FO nodes have a single input port that you use to specify the XML to be converted to PDF. This node expects XML defined using the FO grammar. You can specify a default value, or the value can be dynamic (the output from another node in the XML pipeline, for example). XSL-FO nodes are typically piped to XQuery and XSLT nodes, but they can be piped to any node that outputs XML using the FO grammar.

Output Ports

XSL-FO nodes have a single output port, used to specify what to do with the result of the transformation. Typically you use the **Copy to URL** property to write the output PDF to a file system, but you could also pipe it to a converter that does something with the PDF, or to a Pipeline Output node.

Pipeline and Related Nodes

Including an XML pipeline refers to the process of inserting one XML pipeline inside another – instead of piping a node’s output to an XQuery or XSLT transformation for processing, for example, output is piped to the included XML pipeline. That XML pipeline performs the processes defined by its nodes, and then returns one or more outputs to a node in the including XML pipeline for subsequent processing.

Example

The `getHoldings.pipeline` in the `pipelines\stocks` folder in the `examples` project installed with Stylus Studio uses an included XML pipeline, `retrieveData.pipeline`.

Figure 382 illustrates how an included XML pipeline is represented in the including XML pipeline.

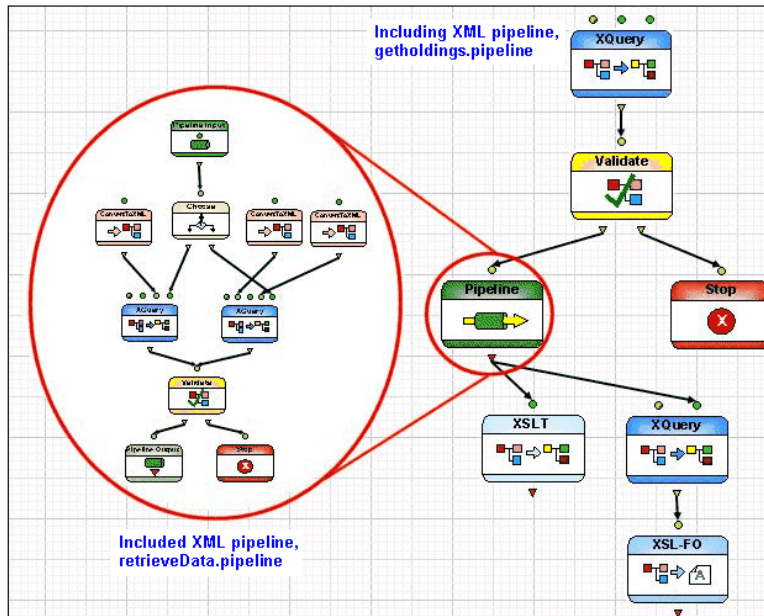


Figure 382. Illustration of an Included Pipeline (retrieveData.pipeline)

Included XML pipelines, as any other documents (XQuery and XSLT, for example), cannot be edited from the including XML pipeline. You must open these documents separately.

Tip To open and edit an included XML pipeline or any other document, double-click its node.

Pipeline Node Input and Output Ports

A Pipeline node displays input and output ports only if the XML pipeline it represents has been defined with Pipeline Input and Pipeline Output nodes, as shown in Figure 392. These nodes allow the included XML pipeline to be connected to the including XML pipeline.

How to Include an XML Pipeline

◆ To include an XML pipeline:

1. Define the XML pipeline (*XML pipeline A*) you want to include in another XML pipeline (*XML pipeline B*).
2. Ensure that you have defined Pipeline Input and Pipeline Output nodes for XML pipeline A (the XML pipeline to be included in XML pipeline B).
3. Drag and drop XML pipeline A into XML pipeline B.
Alternative: Create an empty Pipeline node in XML pipeline B and manually specify the URL for XML pipeline A in its `.pipeline` file property.
4. Connect the input and output ports of the Pipeline node representing XML pipeline A to node ports in XML pipeline B as required.

Validate Nodes

Validate nodes represent an XML Schema document used to validate XML piped to it from another node.

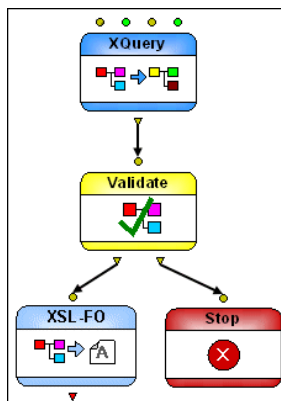


Figure 383. Validate Node in retrieveData.pipeline

You can create a Validate node by dragging an XML Schema document (`.xsd`) and dropping it directly on the XML pipeline canvas, or you can drop an XML Schema document on an existing Validate node.

Using Multiple XML Schemas

A Validate node can represent multiple XML Schemas. You can add additional XML Schemas by simply dragging and dropping the XML Schema document (.xsd) on the Validate node, or you can use the following procedure.

◆ **To add additional XML Schemas to a Validate node:**

1. Display the Properties window if it is not already open (**View > Properties**).
2. Click the Validate node in the XML pipeline diagram.
The properties for the validate node are displayed in the **Properties** window.
3. Click the **XML Schemas** field; click the more button (⋮) when it appears.
The **Select Multiple URLs** dialog box appears.

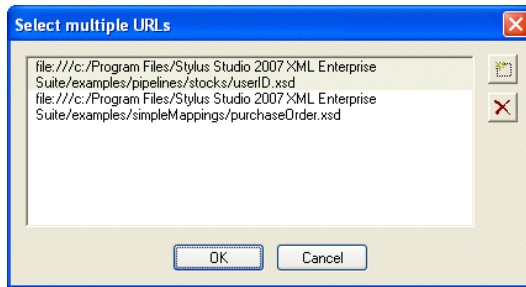


Figure 384. Select Multiple URLs Dialog Box

4. Use the add button (⊕) to display the **Open** dialog box, which you can then use to navigate to the XML Schema you want to add to the Validate node.
5. Click the **OK** button.

Input Port

The Validate node has a single input port that you use to specify the XML you want to be validated by the XML Schemas specified by the Validate node's **XML Schemas** property. You can specify a default value, or the value can be dynamic (the output from another node in the XML pipeline, for example).

Output Ports

Validate nodes have two output ports, named *Output valid* and *Output invalid*, which you use to direct the flow of the XML pipeline. Typically, the *Output valid* port is piped to another transformation in the XML pipeline, while the *Output invalid* port is piped to a node like Stop or Warning.

Choose Nodes

A Choose node uses XPath expressions to evaluate one or more conditions based on its input to direct the flow of XML pipeline processing. The Choose node uses XPath 2.0 to evaluate the XPath expressions defined for it.

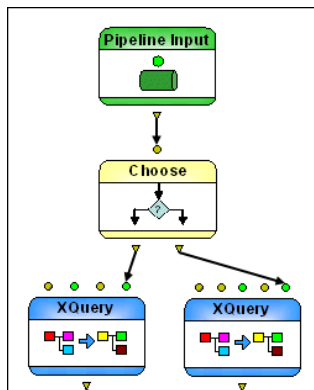


Figure 385. Choose Node in retrieveData.pipeline

Input Ports

The Choose node can have one or more input ports; by default, it has a single input port, named *Input #0*. You use the **XPath #** property to express a “true” condition – that is, a condition that must be met in order for processing to continue. The Choose node in `retrieveData.pipeline`, for example, uses the XPath expression `. castable as xs:double` to check whether or not the user ID it is given as its input is numeric.

The initial context node for each **XPath #** expression is the data input from *Input #0*. For additional input ports (*Input #1*, *Input #2*, and so on), data is available to the XPath expression as `$var1`, `$var2`, and so on.

The Choose node evaluates **XPath#0**. If **XPath#0** is true, it sends data from *Input #0* to *Output #0*. If **XPath#0** is false, it evaluates **XPath#1**. If **XPath#1** is true, it sends data from

Input #0 to *Output #1*, and so on. If none of the **XPath #** expressions is true, the data from *Input #0* is sent to *Output 'no match'* port.

Adding Input Ports

◆ **To add an input port to a Choose node:**

1. Display the Properties window if it is not already open (**View > Properties**).
2. Click the Choose node in the XML pipeline diagram.
3. Change the value for the **Number of inputs** property.
A new input port is added, named *Input #1*.
4. Optionally, add an output port for the new input port.

Output Ports

By default, a Choose node has two output ports:

- *Output#0*, which is enabled when the condition defined by the corresponding property, **XPath #0**, is met (the “true” condition)
- *Output 'no match'*, the default “else” condition, which is enabled if an input evaluates to “false”

You can add additional output ports, which you might want to do if you have added an input port. The output port that corresponds to the “true” condition always makes the first input available to its connecting pipe.

Each output port must have a value specified for it. You can

- Use the **Copy to URL** property to write the output to a file system
- Pipe the output to another node (for more processing if the condition evaluates to “true”, or to a Stop node if the condition evaluates to “false”, for example)
- Or both

Adding Output Ports

◆ To add an output port to a Choose node:

1. Display the Properties window if it is not already open (**View > Properties**).
2. Click the Choose node in the XML pipeline diagram.
3. Change the value for the **Number of choices** property.
A new output port is added, named *Output #1*.

ConvertToXML and ConvertFromXML Nodes

ConvertToXML nodes convert non-XML input (like a CSV file or an EDI message type) to XML; ConvertFromXML nodes convert XML input to a non-XML format. Both nodes use built-in DataDirect XML Converters or user-defined custom XML conversions to convert input.

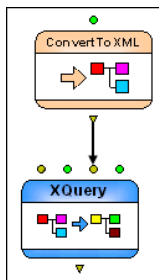


Figure 386. ConvertToXML Node in retrieveData.pipeline


Specifying an XML Converter URL

Both ConvertToXML and ConvertFromXML nodes are defined by specifying an XML Converter URL that evaluates to either a

- Built-in DataDirect XML Converter (converter:CSV or converter:EDI, for example) or a
- User-defined converter (converter:file:///c:/XMLconverters/myConverter.conv, for example)

You can manually type a URL in the node's **XML Converter URL** property, but the XML converter URL syntax can be complex, and it is easy to make errors or to leave settings that you might wish to use unspecified. For example, a completely specified XML Converter URL for the CSV XML Converter might look like this:

```
converter:CSV:newline=cr:sep=|:first=yes:number=yes:collapse=yes
```

The recommended way to specify an XML Converter URL is to select the built-in XML Converter or user-defined custom XML conversion from the **Select XML Converter** dialog box, which you display by clicking the **XML Converter URL** field, and then clicking the more button () when it appears.:

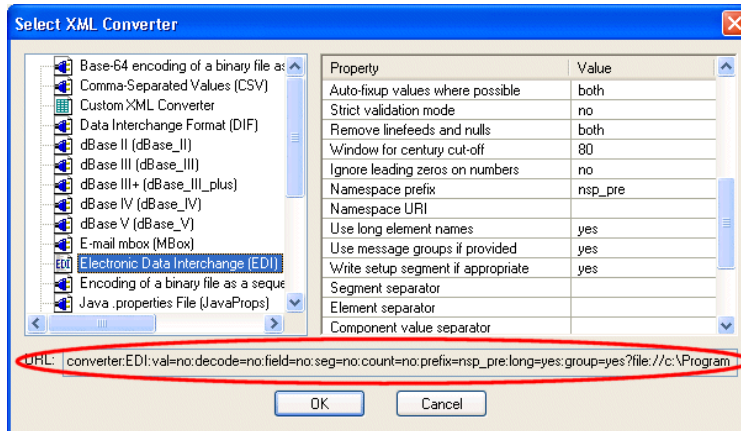


Figure 387. Select XML Converter Dialog Box

The value in the **URL** field changes when you select an XML Converter. Properties and their settings (like `decode=no` in [Figure 387](#)) are displayed only if you change a default value.

Creating a ConvertToXML Node

You can create a ConvertToXML node manually, by dragging the ConvertToXML icon from the **Toolbox** pane and dropping it on the XML pipeline canvas. See [“Create a ConvertToXML Node for order.edi”](#) on page 875 for a description of this procedure.

ConvertToXML nodes are created automatically if an XML Converter URL has been used as a data source for another document represented in the XML pipeline, or if you drag a converted document from the **Project** window and drop it on the canvas. The `Extract_full_order_information` XQuery node in the `order.pipeline` uses two non-XML data sources that convert text and EDI message types to XML for processing. When this XQuery is added to an XML pipeline, its data sources, represented as ConvertToXML nodes, also appear in the XML pipeline.

Input Port

ConvertToXML and ConvertFromXML nodes have a single input port that you use to specify the file to be converted to XML (or vice versa). You can specify a default value, or the value can be dynamic (the output from another node in the XML pipeline, for example).

Output Ports

ConvertToXML and ConvertFromXML nodes have a single output port, used to specify what to do with the result of the transformation. You can

- Use the **Copy to URL** property to write the output to a file system
- Pipe the output to another node (like an XQuery transformation, for example)
- Or both

For More Information

To learn more about built-in DataDirect XML Converters and converter technology, see [“Converting Non-XML Files to XML”](#) on page 215.

Stop and Warning Nodes

Stop and Warning nodes are used to indicate exceptions to or special conditions encountered in an XML pipeline’s execution. They serve a similar purpose, but behave in different ways.

Stop Nodes

[Figure 388](#), for example, shows a Stop node piped to the *Output invalid* output port of the Validate node – if the validation using the XML Schema specified in the Validate node fails, the Stop node aborts XML pipeline processing, and a user-defined error message is

generated. Stop nodes do not have an output port – XML pipeline processing ends if it encounters a Stop node.

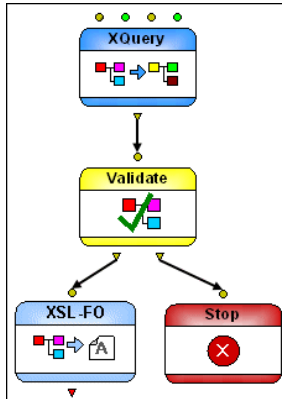


Figure 388. Stop Node in retrieveData.pipeline

Warning Nodes

Warning nodes, like the one shown in [Figure 389](#), do have an output port. When encountered in XML pipeline processing, a Warning node generates the user-defined error message you give to it and pipes the input it is given through to the next node in the XML pipeline.

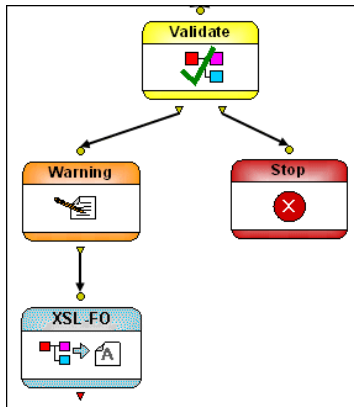


Figure 389. Example Warning Node Implementation

XML Parser Nodes

XML Parser nodes convert text input to an XML document.

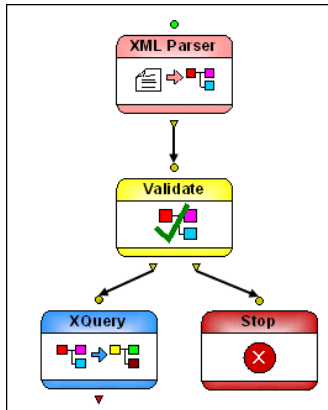


Figure 390. Example XML Parser Node Implementation

Input Port

XML Parser nodes have a single input port that you use to specify the text to be converted to XML. You can specify a default value, or the value can be dynamic (the output from another node in the XML pipeline, for example).

Output Ports

XML Parser nodes have a single output port, used to specify what to do with the result of the transformation. You can

- Use the **Copy to URL** property to write the output to a file system
- Pipe the output to another node (like Validate, for example)
- Or both

An output must be specified in order for the node to be processed.

XML Serializer Nodes

XML Serializer nodes convert an XML document to text. You can use node properties to specify characteristics of the resulting text file, such as whether or not you want it to include an XML declaration, and whether or not to use pretty-print to format it.

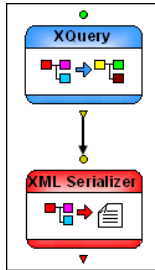


Figure 391. Example XML Serializer Node Implementation

Input Port

XML Serializer nodes have a single input port that you use to specify the XML to be converted to text. You can specify a default value, or the value can be dynamic (the output from another node in the XML pipeline, for example).

Output Ports

XML Serializer nodes have a single output port, used to specify what to do with the result of the transformation. You can

- Use the **Copy to URL** property to write the output to a file system
- Pipe the output to another node
- Or both

Working with the XML Pipeline Diagram

This section describes the features you can use when working with the XML pipeline diagram. This section covers the following topics:

- [“Displaying a Grid”](#) on page 911
- [“Labeling”](#) on page 911
- [“Zoom”](#) on page 912
- [“Edge Style”](#) on page 912
- [“Manipulating Nodes in the Diagram”](#) on page 914
- [“Saving the XML Pipeline Diagram as an Image”](#) on page 914
- [“Labeling XML Pipeline Diagrams”](#) on page 915

Displaying a Grid

By default, Stylus Studio displays a grid on the canvas to help you place nodes in a uniform fashion. You can hide the grid by clicking the **Show Grid** button, or by selecting **XMLPipeline > Show Grid** from the menu or **Show Grid** from the canvas shortcut menu. [Figure 354](#) shows the XML pipeline canvas with the grid displayed.

Tip Use Snap to Grid to have Stylus Studio place nodes precisely along grid line coordinates. This feature is available whether or not you display the grid. See [“Manipulating Nodes in the Diagram”](#) on page 914 for more information.


Labeling

You can label the XML pipeline diagram and individual nodes. Labeling is a useful way to provide documentation for an XML pipeline, especially if you plan to print the diagram.

Tip Open `BookLookup.pipeline`, in the `\pipelines\servlet` folder of the `examples` project installed with Stylus Studio for an example of labels in an XML pipeline diagram.

For more information, see [“Labeling XML Pipeline Diagrams”](#) on page 915.

Zoom

You can zoom the XML pipeline diagram in and out using the zoom slider at the top of the editor () – drag the slider to the right to increase zoom, drag it to the left to decrease zoom. Changing zoom affects both the diagram and the grid (if it is displayed).

The zoom level you select in the XML Pipeline Editor does not affect the size of the nodes if you save the XML pipeline diagram as an image – the default zoom level is always used for saved images.

For more information, see [“Saving the XML Pipeline Diagram as an Image”](#) on page 914.

Edge Style

You can choose from one of three edge styles for the pipes that connect one node to another:

Table 104. Edge Styles

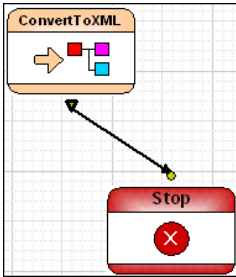
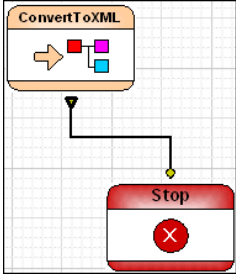
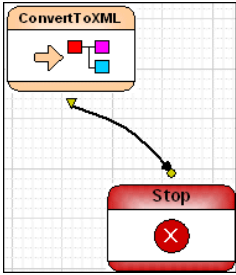
<i>Term</i>	<i>Example</i>
Line	 The diagram shows two nodes on a grid. The top node is labeled 'ConvertToXML' and contains an orange arrow pointing right, a red square, a pink square, and a blue square. The bottom node is labeled 'Stop' and contains a red circle with a white 'X'. A simple black line with an arrowhead at the 'Stop' node connects the two nodes.

Table 104. Edge Styles

Term	Example
Orthogonal	
Spline	

When you change the style, either from the **XMLPipeline** menu or the toolbar, you change the style

- Of any pipes that are currently selected; this includes the pipes associated with any selected nodes
- Of any pipe you create after selecting the new style

◆ **To change the edge style:**

1. Select the pipe, or the node whose pipes, you want to change.
2. Select **XML Pipeline > Edges Style** from the menu.

Alternative: Right-click and select **XML Pipeline > Edges Style** from the shortcut menu.

3. Select the edge style you want from the drop-down menu.

Manipulating Nodes in the Diagram

Once you have added a node to the XML pipeline diagram, there are several ways to manipulate them, as summarized in [Table 105](#). You might want to use these operations to simplify the XML pipeline’s layout, perhaps before saving an image of the diagram. Changing the diagram layout has no effect on the XML pipeline’s definition.

Table 105. Ways to Manipulate Nodes

<i>Action</i>	<i>Description</i>	<i>Methods</i>
Rotate ports	You can rotate the ports on nodes clockwise and counter-clockwise, 90 degrees at a time.	<ul style="list-style-type: none"> ● Rotate (Clockwise or Counter-clockwise) buttons on tool bar ● XML Pipeline > Rotate (Clockwise or Counter-clockwise) on main menu ● Rotate (Clockwise or Counter-clockwise) on shortcut menu
Snap-to-grid	By default, Stylus Studio places nodes where you drop them on the canvas. For a more uniform layout, you can use Snap-to-Grid. When this setting is on, dropped nodes shift (snap) to the closest vertical and horizontal grid axis. Snap-to-Grid is available regardless of whether or not the grid is displayed on the XML pipeline canvas.	<ul style="list-style-type: none"> ● Snap to Grid button on tool bar ● XML Pipeline > Snap to Grid on main menu ● Snap to Grid on shortcut menu

Saving the XML Pipeline Diagram as an Image

You can save a graphical image of your XML pipeline diagram as a JPEG (.jpg) file or as an Extended Meta File (.emf). When you save an XML pipeline as an image, Stylus Studio includes the entire XML pipeline, not just what is currently visible on the XML pipeline canvas.

Stylus Studio uses a standard zoom level when saving an XML pipeline as an image; application zoom level settings are ignored. The grid is captured if it is displayed.

◆ **To save and XML pipeline as an image:**

1. Click the **Save as Image** button on the toolbar.
Alternatives: Select **XMLPipeline > Save as Image** from the menu, or select **Save as Image** from the shortcut menu (right-click).
Stylus Studio displays the **Save As** dialog box.
2. Select the file format (.jpg or .emf) from the **Files of type** drop-down list.
3. Specify a name and location for the file and click the **Save** button. The default name is the name of the XML pipeline; the default location is the project folder in which the XML pipeline has been saved.

Labeling XML Pipeline Diagrams

A *label* is a block of text that you can associate with a node, or with the XML pipeline diagram itself. Labeling can be useful for printed XML pipelines, or any time you need to provide additional documentation about the XML pipeline. Label text is not available in the generated code.

Labels appear as plain text on the canvas until you select one. When you select a label or the node with which it is associated, Stylus Studio displays a line that shows you the node with which the label is associated.

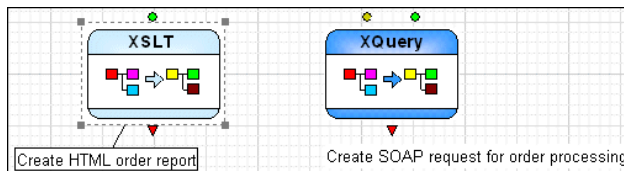


Figure 392. Pipeline Labels

You can create as many labels for a pipeline as you like. Each node, however, can be associated with only one label. You cannot format label text.

Tip For XQuery, XSLT, and other nodes that represent files, use the label to describe the node's action or role in the XML pipeline, and enter the file name in the **Name** property.

◆ **To create a label:**

1. Select the operation you want to label. If you want to label the entire XML pipeline, click the canvas.

2. Select **XML Pipeline > Add Label** from the menu.
Alternative: Right-click and select **Add Label** from the shortcut menu.
A label block appears in the XML pipeline diagram.
3. Type the text for your label.
4. Press Enter.

Debugging an XML Pipeline

You can debug an XML pipeline in Stylus Studio as you would an XQuery or XSLT document – by setting breakpoints, stepping through the diagram, and using features like the **Watch**, **Variables**, and **Call Stack** windows to help troubleshoot your XML pipeline. Debugging tools are available in the toolbar, and in the **Debug** menu.

This section covers the following topics:

- [“Cross-Language Debugging”](#) on page 916
- [“Execution Framework Determines Debugging Support”](#) on page 917
- [“Setting and Removing Breakpoints”](#) on page 917
- [“Running the Debugger”](#) on page 918
- [“Stepping Into a Node”](#) on page 919
- [“Stopping Debug Processing”](#) on page 920

Cross-Language Debugging

An important feature of XML pipeline debugging is Stylus Studio’s support for *cross-language debugging*. Cross-language debugging allows you to set a breakpoint on, say, an XQuery node, and then step into the source XQuery document on which the node is based, debug it, and then step back into the XML pipeline. Cross-language debugging is supported for

- XQuery nodes
- XSLT nodes
- Included pipelines
- Java functions within your XQuery or XSLT code

Execution Framework Determines Debugging Support

The XML pipeline's execution framework determines whether or not you can perform cross-language debugging in your XML pipeline. Processor settings defined in the scenarios for XML pipeline components like XQuery and XSLT have no effect – XML pipeline processor settings are always used.


The following execution framework settings support cross-language debugging:

- Stylus Studio
- Saxonica Saxon and Saxoninca Saxon-SA
- Microsoft .NET

Setting and Removing Breakpoints

You can set breakpoints on any node in an XML pipeline. You cannot set breakpoints on a node's input and output ports.

◆ To set a breakpoint:

1. Select the XML pipeline node on which you wish to set the breakpoint.
2. Click the **Toggle Breakpoint** button () or press F9.
Stylus Studio displays a breakpoint symbol (a large red circle) next to the node.

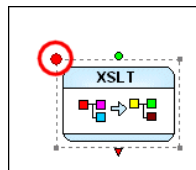



Figure 393. Breakpoint Set on a Node

◆ To remove a breakpoint:

1. Select the XML pipeline node whose breakpoint you wish to remove.
2. Click the **Toggle Breakpoint** button () or press F9.

Running the Debugger

- ◆ To run the debugger, click the Start Debugging button () or click F5:

When the debugger hits a breakpoint you have set, it displays a pause symbol, like the one shown in [Figure 394](#).

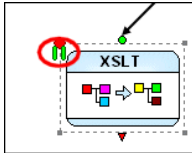


Figure 394. Pause Symbol for a Debugging Breakpoint

When debugging is paused, debugging tools (like those that let you step into and over breakpoints, and toggles for the **Watch**, **Variables**, and **Call Stack** windows) become active. You cannot edit the XML pipeline or alter its scenario properties during debugging, or when the debugger is paused.

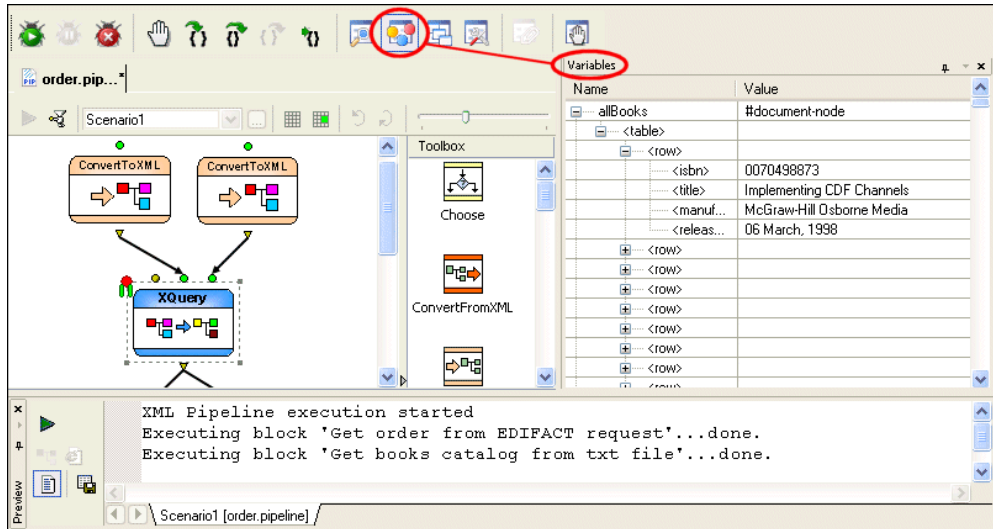



Figure 395. Debugging an XQuery Node in order.pipeline

[Figure 395](#) shows a breakpoint set on the createFullOrder.xquery node in order.pipeline. The **Preview** window shows the XML pipeline's execution log, and we can see in it that the two ConvertToXML nodes have just been processed; this is confirmed

by the presence of the pause symbol on the following XQuery node – it has not yet been processed. The **Variables** window shows the data retrieved from booksXML.txt.

Stepping Into a Node

If we want to take a closer look at the XQuery node as it processes the input, we can step into it, directly from the XML Pipeline Editor by pressing the **Step Into** button () , or by pressing F11. When we step into a node, Stylus Studio opens the document the node represents (in this case, createFullOrder.xquery) in its own editor.

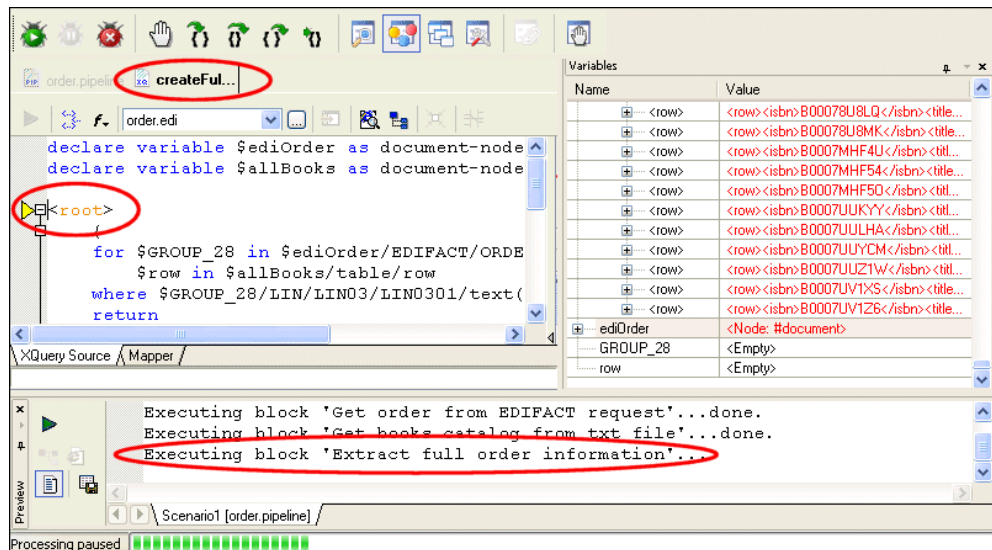





Figure 396. Stepping Into XQuery While Debugging XML Pipeline

When you step into another document from the XML Pipeline Editor, Stylus Studio pauses the debugger on the first instruction in that document. (In XQuery and XSLT, the pause symbol is a yellow triangle. You can step over the instructions, one-by-one, by clicking the **Step Over** button () or pressing F10. You can set breakpoints within this document, as well.

You need to stop debugging before you can make changes to a document.

When the document you have stepped into has completed processing, you are returned to the XML pipeline, and you can continue debugging it.

Stopping Debug Processing

You can stop debug processing of an XML pipeline at any time by clicking the **Stop Debugging** button () or by pressing F8. Similarly, you can pause debugging (when you choose, as opposed to waiting for the debugger to hit a breakpoint) by clicking the **Pause** button () or by selecting **Debug > Pause** from the menu.

Generating Code for an XML Pipeline

Once you have built and tested your XML pipeline in the XML Pipeline Editor and are satisfied that it performs as required, you can generate Java code for it. The generated code includes all the instructions necessary to deploy the XML pipeline in a Java application.

This section covers the following topics:

- [“Processor Settings and Code Generation”](#) on page 920
- [“Code Generation Settings”](#) on page 921
- [“How to Generate Java Code for an XML Pipeline”](#) on page 922
- [“Compiling Generated Code”](#) on page 922
- [“Deploying Generated Code”](#) on page 924

Processor Settings and Code Generation

Settings for the **Processor** properties on the **Deployment** tab of the **Scenario Properties** dialog box influence how Stylus Studio generates Java code for your XML pipeline.

These properties are typically set when you first begin building an XML pipeline, as they also influence how Stylus Studio processes any XQuery, XSLT, XML Schema validation, or FO processing you have specified in your XML Pipeline.

Processors for which Code Generation is Supported

You can generate code for your XML pipeline using any of the following execution frameworks:

- DataDirect XQuery®
- Saxonica Saxon
- Saxonica Saxon-SA

See [“Specifying an Execution Framework”](#) on page 870 for more information.

Code Generation Settings

When you generate code for an XML pipeline, Stylus Studio displays the **Generate Java Code for XML Pipeline** dialog box.

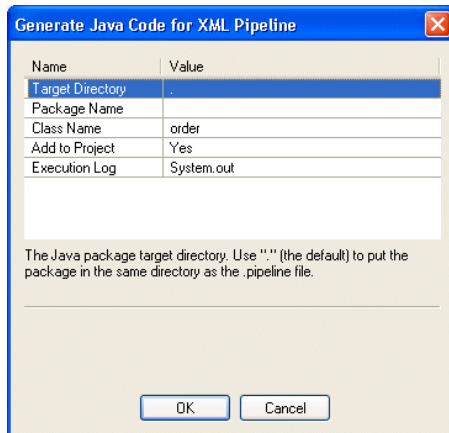


Figure 397. Generate Java Code for XML Pipeline Dialog Box

You use this dialog box to specify

- The target directory in which you want the Java code created. `c:\temp\myPipelineJavaCode`, for example. If the directory you name does not exist, Stylus Studio creates it when you run the Java Code Generation wizard. The default is `.`, which places the generated code in the same directory as the `.pipeline` file.
- Optionally, a package name. If you specify a package name, Stylus Studio uses this name to create a subfolder in the target directory you specify. If you specify *mypackage* as the package name, for example, the generated code is written to `c:\temp\myPipelineJavaCode\mypackage`. (Though optional, it is considered good practice to create a package name.)
- The class name. Stylus Studio also uses the class name for the `.java` file created by the Java Code Generation wizard. For example, if you provide the name *MyClass*, Stylus Studio creates `c:\temp\myPipelineJavaCode\mypackage\MyClass.java`.


In addition, you can specify whether or not you want to

- Add the generated code to the current project
- Write an execution log file when the Java class runs

All of these options are selected by default.

How to Generate Java Code for an XML Pipeline

◆ **To generate Java code for an XML Pipeline:**

1. Make sure properties on the **Deployment** tab of the **Scenario Properties** dialog box for the XML pipeline are set appropriately. See [“Processor Settings and Code Generation”](#) on page 920 for more information.
2. Click the **Generate Java Code** button () on the XML Pipeline Editor toolbar.
Alternative: Select **XML Pipeline > Generate Java Code** from the Stylus Studio menu. The **Generate Java Code for XML Pipeline** dialog box appears.

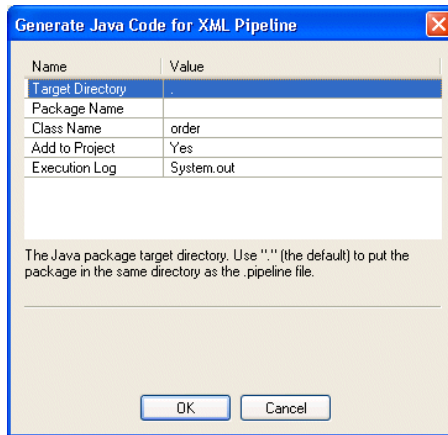


Figure 398. Java Code Generation Dialog Box

3. Specify the settings you want for the target directory, package and class names, and so on. See [“Code Generation Settings”](#) on page 921 if you need help with this step.
4. Click **OK**.
Stylus Studio generates Java code for the XML pipeline. When the code generation is complete, the resulting file (*classname.java*) is opened in the Stylus Studio Java Editor.

Compiling Generated Code

The deployer automatically puts the JAR files required to compile the generated Java code in the Stylus Studio project classpath. JAR files are in the \bin directory where you installed Stylus Studio.

How to Compile and Run Java Code in Stylus Studio

In order to compile Java code, the JDK must be installed on your machine and configured in Stylus Studio. Click **Tools > Options > Java Virtual Machine** to configure the JDK.

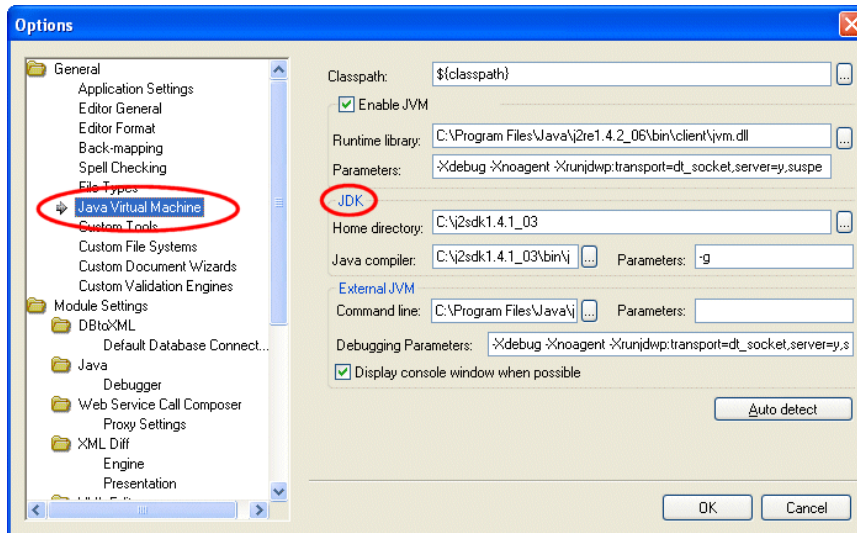



Figure 399. Configuring the JDK in Stylus Studio

◆ To compile Java code in Stylus Studio:

1. Make sure the Java Editor is the active window.
2. Click the **Compile** button ().

Alternatives: Press **Ctrl + F7**, or select **Java > Compile** from the Stylus Studio menu. Stylus Studio compiles the Java code. Results are displayed in the **Output** window.

Troubleshooting Compiling Inside Stylus Studio

If you have trouble compiling Java code in Stylus Studio

1. Remove the existing Stylus Studio project classpaths from the **Project Classpath** dialog box (select **Project > Set Classpath** from the menu).
2. Generate code again, which causes Stylus Studio to respecify the project classpaths.

Compiling Java Code Outside Stylus Studio

If you want to compile the Java code generated for your XML pipeline outside Stylus Studio, you will need to manually set your classpath to include the JAR files listed at the top of the generated `.java` file.

Running Java Code in Stylus Studio

◆ **To run Java code in Stylus Studio:**

1. Make sure the Java Editor is the active window.
2. Click the **Run** button ().

Alternatives: Press `Ctrl + F5`, or select **Java > Run** from the Stylus Studio menu.

If the code has not been compiled, Stylus Studio displays a prompt asking if you want to compile the code now. Otherwise, Stylus Studio runs the Java code. Results are displayed in the **Output** window.

Deploying Generated Code

If your XML Pipeline uses built-in DataDirect XML Converters™ – to convert CSV or EDI to XML, for example – you need to purchase licenses for the DataDirect XML Converters you wish to use if you wish to deploy your code in any environment on a machine (such as a test or application server) that does not have a license for the DataDirect XML Converters. Licenses for DataDirect XML Converters are purchased separately from Stylus Studio 2007 XML Enterprise Suite.

Similarly, if you use DataDirect XQuery® in your XML pipeline, you must acquire additional licences if you wish to deploy the XML pipeline application.

Write Stylus Studio at stylusstudio@stylusstudio.com, or call 781.280.4488 for more information.

XML Pipeline Node Properties Reference

This section contains reference information for XML pipeline node properties, including their input and output ports. This section is organized as follows:

- “Choose Node Properties” on page 925
- “ConvertFromXML Node Properties” on page 927
- “ConvertToXML Node Properties” on page 928

- [“Pipeline Node Properties”](#) on page 929
- [“Pipeline Input Node Properties”](#) on page 930
- [“Pipeline Output Node Properties”](#) on page 931
- [“Stop Node Properties”](#) on page 931
- [“Validate Node Properties”](#) on page 932
- [“Warning Node Properties”](#) on page 933
- [“XML Parser Node Properties”](#) on page 934
- [“XML Serializer Node Properties”](#) on page 935
- [“XQuery Node Properties”](#) on page 936
- [“XSL-FO Node Properties”](#) on page 937
- [“XSLT Node Properties”](#) on page 938

Choose Node Properties

Input Port

Choose nodes can have as many input ports as you specify.

Table 106. Choose Input Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed in the port’s tooltip. Default value is Input#0. This number is incremented by one for each additional input port (Input#1, Input#2, and so on). Not editable.
DataType	The port’s datatype. Default value is node(). Other values are available in a drop-down list.
Default Value	The default value for the input. Can be blank (this is the default).

Node

Table 107. Choose Node Properties

<i>Property</i>	<i>Description</i>
Name	The name you want to appear in the Choose node's tooltip. Default value is Choose.
Number of Inputs	The number of input ports you want the Choose node to have.
Number of Choices	By default, a Choose node has two choices – one, which you specify, and the “else”, which is implicit. You can use this property to specify additional choices.
XPath #0	The XPath expression used to define the choices in the Choose node. There is one XPath# property for each choice.

Output Port

Each Choose node has at least two output ports – Output#0, and Output 'no match'. It will have other output ports (Output#1, Output#2, and so on) if other choices have been defined for the Choose node.

Table 108. Choose Output Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the output port; not editable.
DataType	The port's datatype. Default value is node().
Copy to URL	The URL to which you want the output passed. Can be left blank.

ConvertFromXML Node Properties

Input Port

Table 109. ConvertFromXML Input Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the input port; not editable.
DataType	The port's datatype. Default value is node.
Default Value	The default value for the input. Can be blank (this is the default).

Node

Table 110. ConvertFromXML Node Properties

<i>Property</i>	<i>Description</i>
Name	The name you want to appear in the ConvertFromXML node's tooltip. Default value is ConvertFromXML.
XML Converter URL	The URL of the DataDirect XML Converter (converter:CSV?, for example) or of the user-defined custom XML conversion (converter:file://c:\myFiles\myConverter.conv?, for example) you want to use to convert XML input to some other format.

Output Port

Table 111. ConvertFromXML Output Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the output port; not editable.
DataType	The port's datatype. Default value is text.
Copy to URL	The URL to which you want the output passed. Can be left blank.

ConvertToXML Node Properties

Input Port

Table 112. ConvertToXML Input Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the input port; not editable.
Data Type	The port's datatype. Default value is node.
Default Value	The default value for the input. Can be blank (this is the default).

Node

Table 113. ConvertToXML Node Properties

<i>Property</i>	<i>Description</i>
Name	The name you want to appear in the ConvertToXML node's tooltip. Default value is ConvertToXML.
XML Converter URL	The URL of the DataDirect XML Converter (converter:CSV?, for example) or of the user-defined custom XML conversion (converter:file://c:\myFiles\myConverter.conv?, for example) you want to use to convert input to XML.

Output Port

Table 114. ConvertToXML Output Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the output port; not editable.
Data Type	The port's datatype. Default value is node.
Copy to URL	The URL to which you want the output passed. Can be left blank.

Pipeline Node Properties

A Pipeline node does not display input and output ports unless the included pipeline it represents (specified in the `.pipeline File` property) has Pipeline Input and Pipeline Output nodes defined for it.

Input Port

Table 115. Pipeline Input Port Properties

<i>Property</i>	<i>Description</i>
Name	The name of the Pipeline Input node that is defined in the included pipeline; not editable.
DataType	The port's datatype. Default value is <code>text</code> .
Default Value	The default value for the input. Can be blank (this is the default).

Node

Table 116. Pipeline Node Properties

<i>Property</i>	<i>Description</i>
Name	The name you want to appear in the Pipeline node's tooltip. Default value is <code>Include Sub-Pipeline</code> .
<code>.pipeline File</code>	The URL of the <code>.pipeline</code> file represented by the Pipeline node.

Output Port

Table 117. Pipeline Node Output Port Properties

<i>Property</i>	<i>Description</i>
Name	The name of the Pipeline Output node that is defined in the included pipeline; not editable.
DataType	The port's datatype. Default value is <code>text</code> .
Copy to URL	The URL to which you want the output passed. Can be left blank.

Pipeline Input Node Properties

Pipeline Input nodes have no input port.

Node

Table 118. Pipeline Input Node Properties

<i>Property</i>	<i>Description</i>
Name	The name you want to appear in the Pipeline Input node's tooltip. Default value is Pipeline Input.
DataType	The node's datatype. Default value is any.
Default Value	The literal value or URL for the document or file you want to use as the included pipeline's main input.

Output Port

Table 119. Pipeline Input Output Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the output port; not editable.
DataType	The port's datatype. Default value is any.

Pipeline Output Node Properties

Pipeline Output nodes do not have an output port.

Input Port

Table 120. Pipeline Output Input Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the input port; not editable.
DataType	The port's datatype. Default value is any.

Node

Table 121. Pipeline Output Node Properties

<i>Property</i>	<i>Description</i>
Name	The name you want to appear in the Pipeline Output node's tooltip. Default value is Pipeline Output.

Stop Node Properties

Stop nodes do not have an output port.

Input Port

Table 122. Stop Node Input Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the input port; not editable.
DataType	The port's datatype. Default value is any.

Node

Table 123. Stop Node Properties

<i>Property</i>	<i>Description</i>
Name	The name you want to appear in the Stop node's tooltip. Default value is Stop.
Message	The message you want written to output with the Stop node is processed. The default is Error.

Validate Node Properties

Input Port

Table 124. Validate Node Input Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the input port; not editable.
DataType	The port's datatype. Default value is node.
Default Value	The default value for the input. Can be blank (this is the default).

Node

Table 125. Validate Node Properties

<i>Property</i>	<i>Description</i>
Name	The name you want to appear in the Validate node's tooltip. Default value is Validate operator.
XML Schemas	The URL of the XML Schema you want to use to validate the input. You can specify more than one XML Schema.

Output Port

The Validate node has two output ports – one (named `Output valid`) used to direct input if the XML Schema validation passes, the other (named `Output invalid`) used if the input is invalid.

Table 126. Validate Node Output Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the <code>Output valid</code> and <code>Output invalid</code> ports; not editable.
DataType	The port's datatype. Default value is <code>node</code> .
Copy to URL	The URL to which you want to copy the output.

Warning Node Properties

Input Port

Table 127. Warning Node Input Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the input port; not editable.
DataType	The port's datatype. Default value is <code>any</code> .

Node

Table 128. Warning Node Properties

<i>Property</i>	<i>Description</i>
Name	The name you want to appear in the Warning node's tooltip. Default value is <code>Warning</code> .
Message	The message you want written to output with the Warning node is processed. The default is <code>Warning</code> .

Output Port

Table 129. Warning Node Output Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the output port; not editable.
DataType	The port's datatype. Default value is any.
Copy to URL	The URL to which you want to copy the output when a warning condition is encountered.

XML Parser Node Properties

Input Port

Table 130. XML Parser Node Input Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the input port; not editable.
DataType	The port's datatype. Default value is text.
Default Value	The default value for the input. Can be blank (this is the default).

Node

Table 131. XML Parser Node Properties

<i>Property</i>	<i>Description</i>
Name	The name you want to appear in the XML Parser node's tooltip. Default value is XML Parser.

Output Port

Table 132. XML Parser Node Output Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the output port; not editable.
Data Type	The port's datatype. Default value is any.
Copy to URL	The URL to which you want to copy the output XML parser output.

XML Serializer Node Properties

Input Port

Table 133. XML Serializer Node Input Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the input port; not editable.
Data Type	The port's datatype. Default value is node.
Default Value	The default value for the input. Can be blank (this is the default).

Node

Table 134. XML Serializer Node Properties

<i>Property</i>	<i>Description</i>
Name	The name you want to appear in the XML Serializer node's tooltip. Default value is XML Serializer.
format-pretty-print	Whether or not you want to format the output using pretty-print (indenting nodes). The default is False.
xml-declaration	Whether or not you want the output to include an XML declaration. The default is true.
encoding	The type of encoding you want to specify for the output. The default is utf-8 .

Output Port

Table 135. XML Serializer Node Output Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the output port; not editable.
DataType	The port's datatype. Default value is text.
Copy to URL	The URL to which you want to copy the output XML parser output. Can be left blank.

XQuery Node Properties

Input Port

An XQuery node has one input port by default. Additional input ports are based on external variables defined in the XQuery.

Table 136. XQuery Node Input Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the input port; not editable.
Data Type	Present for input ports corresponding to external variables. Initialized with variable's data type.
Default Value	The default value for the input. Can be blank.

Node

Table 137. XQuery Node Properties

<i>Property</i>	<i>Description</i>
Name	The name you want to appear in the XQuery node's tooltip. Default value is XQuery operator.
.xquery file	The URL of the XQuery file you want this node to represent.
DB Connections	Reserved for future use.

Output Port

Table 138. XQuery Node Output Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the output port; not editable.
DataType	The port's datatype. Default value is any.
Copy to URL	The URL to which you want to copy the XQuery output. Can be left blank.

XSL-FO Node Properties

Input Port

Table 139. XSL-FO Node Input Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the input port; not editable.
DataType	The port's datatype. Default value is node.
Default Value	The default value for the input. Can be blank.

Node

Table 140. XSL-FO Node Properties

<i>Property</i>	<i>Description</i>
Name	The name you want to appear in the XSL-FO node's tooltip. Default value is FO operator.

Output Port

Table 141. XSL-FO Node Output Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the output port; not editable.
DataType	The port's datatype. Default value is binary.
Copy to URL	The URL to which you want to copy the XSL-FO output.

XSLT Node Properties

Input Port

An XSLT node has one input port by default. Additional input ports are based on parameters defined in the XSLT.

Table 142. XSLT Node Input Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the input port; not editable.
Default Value	The default value for the input. Can be blank.

Node

Table 143. XSLT Node Properties

<i>Property</i>	<i>Description</i>
Name	The name you want to appear in the XSLT node's tooltip. Default value is XSLT operator.
.xsl file	The URL of the XSLT file you want this node to represent.
Base URL	The URL you want to use to resolve hyperlinks and images in the output. This value defaults from the XSLT scenario properties if it was specified.

Output Port

Table 144. XSLT Node Output Port Properties

<i>Property</i>	<i>Description</i>
Name	The name displayed for the output port; not editable.
DataType	The port's datatype. Default value is any.
Copy to URL	The URL to which you want to copy the XSLT output. Can be left blank.

Chapter 13 Publishing XML Data

This chapter describes how to use the Stylus Studio XML Publisher to create XSLT or XQuery code that generates HTML+CSS or XSL-FO reports based on XML data.



Support for XML Publisher is available only in Stylus Studio XML Enterprise Suite.



Watch it! You can view a video demonstration of this feature by clicking the television icon or by clicking this link: [watch the Introduction to the XML Publisher video](#).

A complete list of the videos demonstrating Stylus Studio's features is here: http://www.stylusstudio.com/xml_videos.html.

This chapter covers the following topics:

- “The Stylus Studio XML Publisher” on page 942
- “Building an XML Publisher Report” on page 943
- “Working with Data Sources” on page 945
- “Adding Data to a Report” on page 959
- “Working with Report Components” on page 966
- “Generating Code for an XML Publisher Report” on page 989
- “Example: Building an XML Publisher Report” on page 993
- “Properties Reference” on page 999

Note The examples in this chapter rely on the `books.xml` and `videos.xml` documents that are part of the `examples` project installed with Stylus Studio. These documents are in the `query` and `VideoCenter` project folders, respectively.

The Stylus Studio XML Publisher

The Stylus Studio XML Publisher is a visual editor and code generator that helps you create XSLT or XQuery that transforms XML into HTML+CSS or XSL-FO reports.

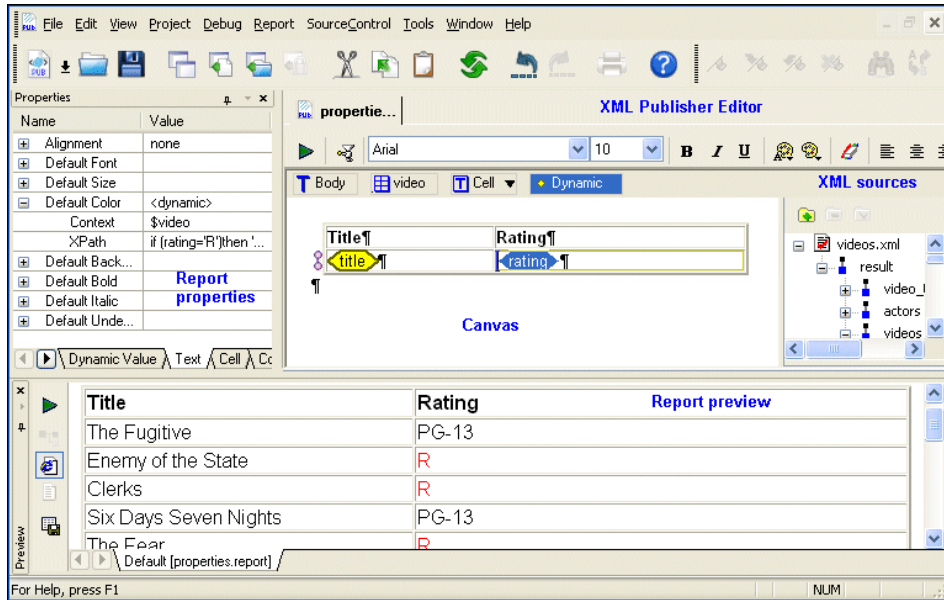


Figure 400. XML Publisher and Report Preview

You use the XML Publisher to select the data sources for your report, to build and format the report using the desired data, and to preview a sample report before you generate the XSLT or XQuery code.

Parts of the XML Publisher Editor

The XML Publisher Editor has three main parts:

- The *XML Publisher canvas*. You use the canvas to format your report and identify its contents.
- The *data sources panel*. You use the data sources panel to identify the XML sources (relational tables and XML documents, for example) you want to include in your finished report. You select the data you want in your report by dragging the nodes that represent the data from the data sources panel and dropping them on the XML Publisher canvas.

- The *Properties window*. You use the **Properties** window to fine-tune settings that affect the finished report's format and contents. For example, you can use XPath expressions to specify formatting based on results returned by the XPath expression against a given document node.

In addition, Stylus Studio displays the **Preview** window when you preview a report.

Building an XML Publisher Report

This section describes how to build an XML Publisher report. It contains the following sections:

- [“Process Summary”](#) on page 943
- [“How to Create an XML Publisher Report”](#) on page 944
- [“The XML Publisher Canvas”](#) on page 944

Process Summary

The process of building an XML Publisher report involves the following basic steps:

1. Create a new XML Publisher report document (**File > New > XML Report**).
2. Add the source documents you want to use for the report's content. See [“Working with Data Sources”](#) on page 945.
3. Add the components you want in your report – data, as well as formatting constructs like lists and tables – to the XML Publisher canvas. See [“Adding Data to a Report”](#) on page 959.
4. Optionally, format the data you have added to the XML Publisher canvas. See [“Formatting Components”](#) on page 984.
5. Preview the report; adjust data and formatting as required.

Once you are satisfied with the XML Publisher report, you can generate XQuery or XSLT code for the report, choosing your desired report format (HTML or XSL-FO).

Each of these steps is described in greater detail in the following sections.

How to Create an XML Publisher Report

- ◆ To create an XML Publisher report, select **File > New > XML Report from the Stylus Studio menu.**

The XML Publisher Canvas

When you create an XML Publisher report, Stylus Studio starts the XML Publisher and displays a new report (untitled.report). The canvas representing the report is empty, as shown here.

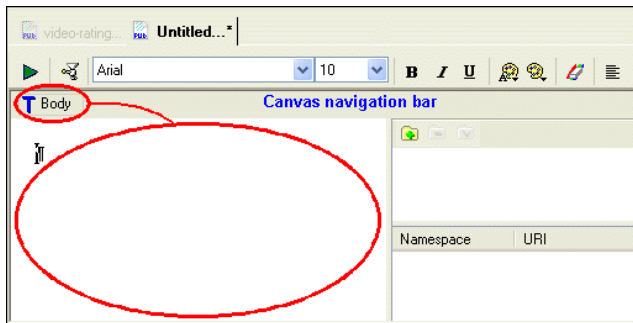


Figure 401. Empty Canvas in a New XML Publisher Report

The canvas represents the report’s body, and all of the work you do when building a report – inserting XML data, tables, text and text blocks, and conditional expressions – takes place within the context of the body.

A useful way to think of the body is as the `<body>` tag in an HTML document – everything that you do on the report canvas would be described between the `<body>` and `</body>` tags in an HTML document. This metaphor is represented by the **Body** glyph (`TBody`) in the canvas navigation bar at the top of the canvas – everything you subsequently add to the report canvas will be represented as a *child* of the report’s body.

See “[How Data is Represented on the Canvas](#)” on page 961 and “[More About the Navigation Bar](#)” on page 963 for more information.

Working with Data Sources

You can use any of the following as data sources for building XML Publisher reports:

- XML documents
- XML Schema or DTD
- Relational database tables
- EDI and flat files like CSV converted to XML using one of the DataDirect XML Converters
- Web services

This section covers the following topics:

- [“How Data Sources are Represented in XML Publisher”](#) on page 945
- [“Adding a Data Source”](#) on page 946
- [“Specifying a Default Data Source”](#) on page 947
- [“Data Source Required for XSLT”](#) on page 948
- [“Using XML Schema or DTD as a Data Source”](#) on page 948
- [“Grouping Data”](#) on page 950

How Data Sources are Represented in XML Publisher

When you add a data source to the data sources panel, Stylus Studio displays the schema representation, or the data model, for that source. The following illustration shows how the `videos.xml` document from the `VideoCenter` folder in the `examples` project appears after it has been added as a data source:

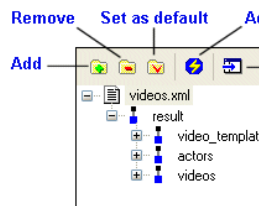


Figure 402. XML Document in the Data Sources Panel

Working with Namespaces

If the document you have selected as a source uses a namespace prefix, Stylus Studio displays the prefix and the URI at the bottom of the data sources panel, as shown in [Figure 403](#).

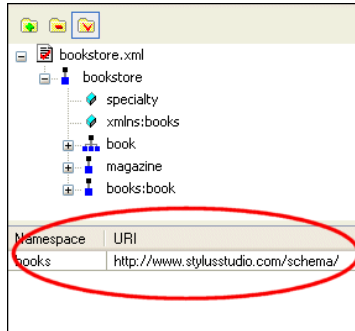



Figure 403. Namespace Prefix and URI

You can edit the value in the **Namespace** field, which can simplify the process of typing XPath expressions you might use when defining a property. For example, you could change books to simply b to shorten and simplify XPath expressions. See [“Example: Using Context and XPath Sub-Properties to Format Text”](#) on page 981 to learn more about using XPath expressions when building XML Publisher reports.

Adding a Data Source

◆ **To add a data source you can**

- Drag a document or file from the **Project** or **File Explorer** windows and drop it on the data sources panel.
- Drag a relational database table from the **File Explorer** window and drop it on the data sources panel.
- Click the **Add Data Source** button () on the data sources panel and use the **Open** dialog box to navigate to the desired data source.

You can use multiple data sources for an XML Publisher report.

Specifying a Default Data Source

Stylus Studio uses the first data source you add as the *default data source*. The default data source is specified as the **Main input** for XQuery scenarios, and as the **Source XML URL** for XSLT scenarios. In other words, when you generate XQuery or XSLT for your XML Publisher report, the default data source is automatically specified in the scenario properties, as shown in [Figure 404](#).

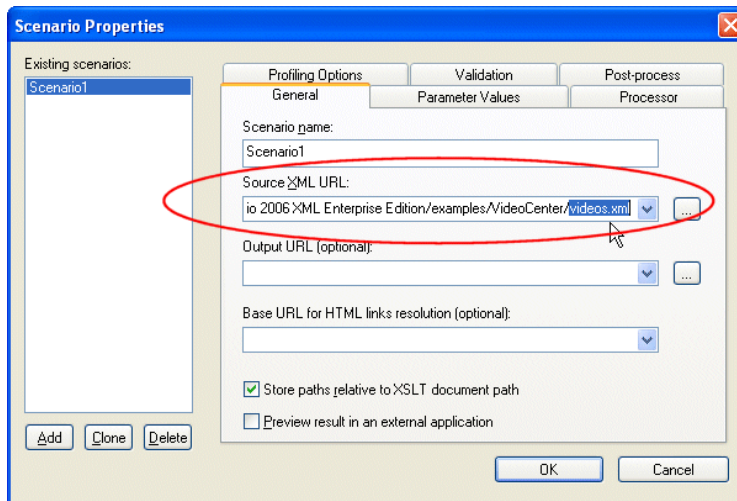


Figure 404. Default Data Source Used in Generated XSLT

The red check on the document icon (see [Figure 402](#)) indicates a data source's default status. You can specify any data source you add as the default data source.

◆ To specify the default data source:

1. In the data sources panel, select the data source you want to specify as the default (books.xml, in [Figure 405](#), for example).

2. Click the **Set As Default** button ().

The data source is set as the new default, as indicated by the red check.

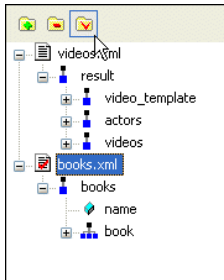


Figure 405. Setting a Different Default Data Source

Data Source Required for XSLT

While it is considered good practice to specify a data source regardless of whether you are planning to generate XQuery or XSLT, a data source is required only for XSLT. In addition, if your XML Publisher report has multiple data sources, one of them must be designated as the default data source. See [“Specifying a Default Data Source”](#) on page 947 for more information.

See [“Sources”](#) on page 990 to learn more about how additional sources are treated by Stylus Studio when generating XQuery and XSLT from XML Publisher.

Using XML Schema or DTD as a Data Source

If you use XML Schema or DTD documents as sources for XML Publisher reports, you need to

- Choose the element from the XML Schema or DTD you want to use as the root element
- Associate the XML Schema or DTD with an XML instance

This section describes how to perform these procedures.

Choosing a Root Element

When you add an XML Schema or a DTD to the data sources panel, Stylus Studio displays the **Choose Root Element** dialog box, shown in [Figure 406](#).

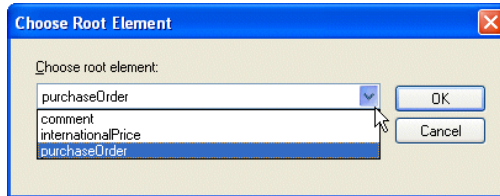


Figure 406. Choose Root Element Dialog Box

The **Choose root element** drop-down list displays all the child elements of the XML Schema or DTD document you selected as a data source. Select the element you want to use as the document root and click **OK**.

Associating an XML Instance with the Schema

Before you can preview or generate code for an XML Publisher report, you need to associate an XML document, referred to as an *XML instance*, with any XML Schema or DTD documents you are using as source documents.

When you click the **Preview** or **Generate** buttons on the XML Publisher toolbar, if you have not already associated an XML instance with the schema you are using as data sources, Stylus Studio displays the **Associate with XML Instance** dialog box, shown in [Figure 406](#).

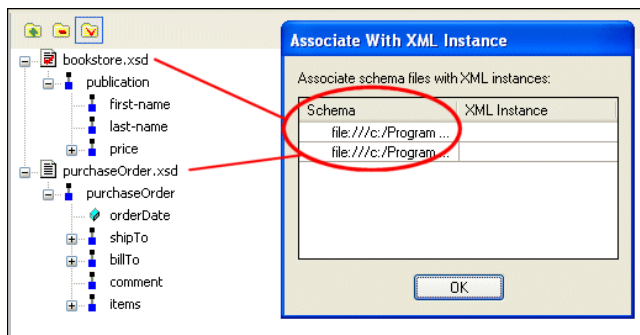


Figure 407. Schema Instance Dialog Box

Each entry in the **Schema** field represents an XML Schema or DTD document used as a data source. To associate it with an XML instance:

1. Click the **XML Instance** field.
Stylus Studio displays the **Open** dialog box.
2. Choose the XML document you want to use as the XML instance and click the **Open** button.
The **Open** dialog box closes; the URL for the file you selected appears in the **XML Instance** field.
3. Click **OK**.
Stylus Studio previews the XML Publisher report, or begins the code generation process.

Grouping Data

The ability to group data from one or more data sources is a common requirement for many reports. For example, given the `videos.xml` file, you might want to create a list of actors that shows all the movies in which he or she has starred.

Stylus Studio facilitates grouping using a feature that allows you to create a relationship between different data sources (between `books.xml` and `catalog.xml`, for example), or between different data islands within the same source (between two nodes in `videos.xml`, for example).

This section describes the relationship feature in XML Publisher and how to use it to perform grouping.

What is a Relationship?

A *relationship* is a link between two nodes in one or more data sources that allows you to compare the values of those nodes. For example, in `videos.xml`, you might want compare the value of the `id` attribute of the `actor` element with the value of `actorRef` element, and then perform some action when those values are equal.

Comparison operations you can define for a relationship are

- Equal
- Not equal
- Less than
- Greater than

- Less than or equal to
- Greater than or equal to

When you create a relationship in XML Publisher, you are defining the inner and outer loops of the for-each statements in your XSLT or XQuery code that will be used to perform the grouping (`xs1:for-each` in XSLT; `FLWOR` instructions in XQuery). The order of the nodes you select determines the order in which the outer and inner loops are created:

- The first node you select is used to define the outer loop
- The second node you select is used to define the inner loop

Creating a Relationship

Once you have added one or more data sources to the data sources panel, you can create a relationship between nodes within the same data source, or across data sources.

◆ To create a relationship:

1. Add the data source(s) you require for your report. See [“Adding a Data Source”](#) on page 946 if you need help with this step.
2. Select the node you want to use to define the outer loop in the XQuery or XSLT that will be used to create the report.

The **Add Relationship** button becomes active, as shown here:

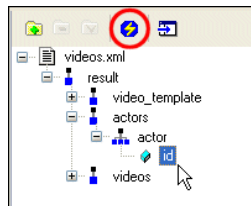


Figure 408. Add Relationship Button for XML Publisher

3. Click the **Add Relationship** button.

The **Create Relationship** dialog box appears. The tree for the document appears in the **Link From** field. The document that appears in the **Link To** field depends on how many data sources you added in [Step 1](#).

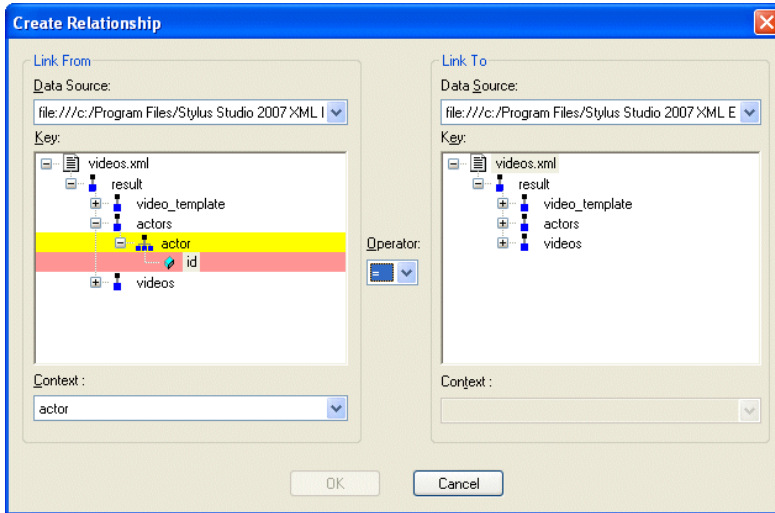


Figure 409. Create Relationship Dialog Box

By default, Stylus Studio sets the node's context using the first repeating element in the selected node's hierarchy – including the selected node itself. In this example, we selected the `id` attribute of the `actor` element, so the `actor` repeating element is used to set the context for this loop.

4. Optionally, change the key node and context.
5. In the **Link To** field, select the node you want to use to define the inner loop in the XQuery or XSLT that will be used to create the report. If you defined more than one data source in [Step 1](#), you can change the data source in the **Data Source** drop-down list.

In this example, we selected `videos/video/actorRef` repeating element.

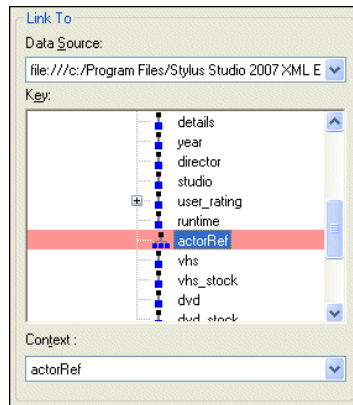


Figure 410. Second Node in a Relationship

6. Optionally, change the context of the node you selected in [Step 5](#).
7. Choose the comparison operator you want to use to define this relationship from the **Operator** drop-down list.
8. Click **OK**.

The relationship you just defined appears in the data sources pane of the XML Publisher Editor.

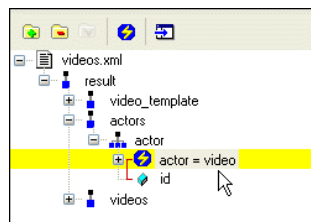


Figure 411. Relationship Defined as a Data Source

If you expand the relationship node, you see the graphic representation of the join formed between the actors/actor/id node and the videos/video/actorRef node.

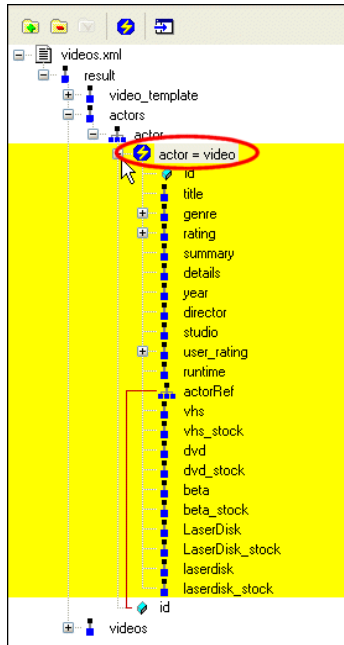


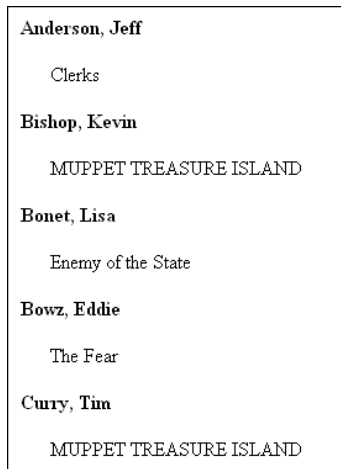
Figure 412. Join Displayed as a Data Source

You can now use this relationship as a data source in XML Publisher. See [“Example – Using a Relationship in a Report”](#) on page 954 for more information.

Example – Using a Relationship in a Report

This example describes how to build a simple report in XML Publisher, shown here, that lists actors and the movies they have appeared in. The information for this report is based on the data in videos.xml, in the VideoCenter folder in the Stylus Studio examples project.

Specifically, it matches the `id` attribute in the `actors/actor` element with the `videos/video/actorRef` element.



Anderson, Jeff	Clerks
Bishop, Kevin	MUPPET TREASURE ISLAND
Bonet, Lisa	Enemy of the State
Bowz, Eddie	The Fear
Curry, Tim	MUPPET TREASURE ISLAND

Figure 413. Example Report

◆ **To create the example report:**

1. Click **File > New > XML Report** to open the XML Publisher Editor.
2. Drag `videos.xml` from the VideoCenter folder in the Stylus Studio examples project and drop it on the data sources panel in the XML Publisher Editor.
Stylus Studio displays a tree representing the `videos.xml` document.
3. Expand the `actors` node by selecting it and pressing the * key on your number pad.
4. Select the `id` attribute.
5. Click the **Add Relationship** button.

The **Create Relationship** dialog box appears.

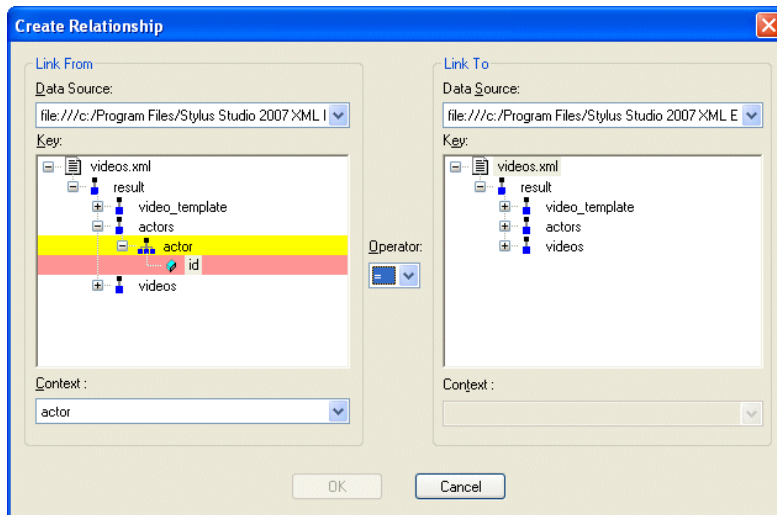


Figure 414. Create Relationship Dialog Box

6. In the **Link To** field, expand the videos node and select the actorRef repeating element.
7. Since we want the loop on the video element (to locate all movies with a matching actorRef and id), change the value in the **Context** field to video (also a repeating element).

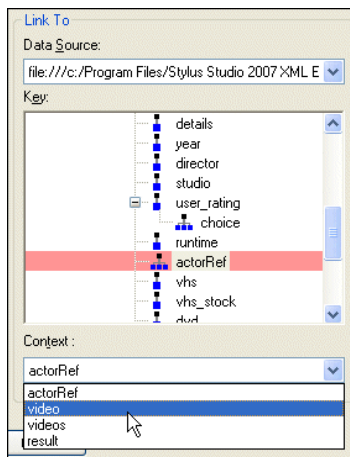


Figure 415. Changing the Context of the Target Node

8. Click **OK**.

The data source defined by the relationship we just created between the actors and videos nodes appears in the data sources pane.

9. Drag the newly defined data source from the data sources pane and drop it on the XML Publisher canvas.

Stylus Studio creates two loops.

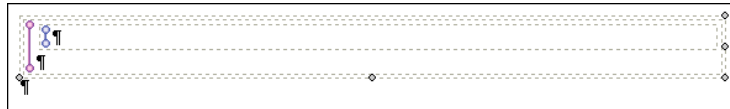


Figure 416. Repeating Loops in the XML Publisher Canvas

If you place the mouse over the outer loop, the tool tip displays the XPath – `/result/actors/actor`; similarly the XPath for the inner loop is `/result/videos/video[$actor/@id./actorRef]`.

Now that the context for the loops has been defined, we next need to specify the data we want to display.

10. From the data sources panel, drag the `videos/video/title` element and drop it in the inner loop.
11. From the data sources panel, drag the `actors/actor` element and drop it in the outer loop. Select **Insert Value** from the pop-up menu.

When you have finished, your XML Publisher canvas should look like this:



Figure 417. XML Publisher Before Formatting

- Click the **Preview** button (▶), and save the file when prompted.
Before formatting, the report looks like this:

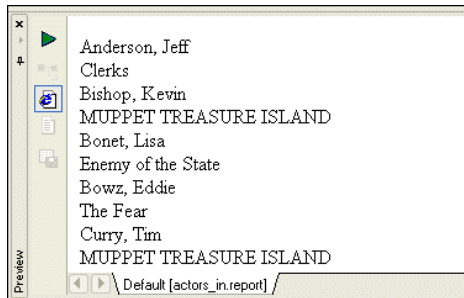


Figure 418. Draft Report

All the information is there, but the report is hard to read.

- Select the **(actor)** glyph in the XML Publisher canvas and click the **Bold** button on the XML Publisher tool bar. Add a carriage return (press Enter) after the glyph.
- Use the space bar to indent the **title** glyph. Add a carriage return (press Enter) after the glyph.
- Preview the report again.

We now have a report that resembles the one shown in [Figure 413](#).

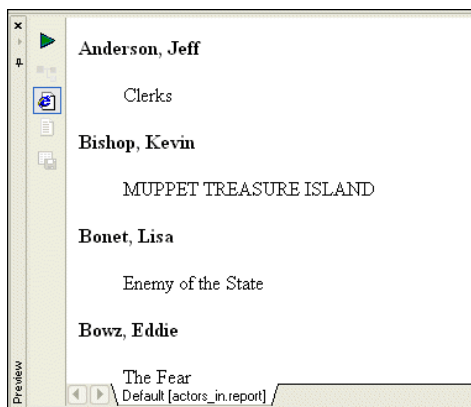


Figure 419. Final Report

Deleting a Relationship

You can delete the relationships you have defined as data sources for XML Publisher reports just as you would any other data source.

◆ **To delete a relationship:**

1. Select the relationship you want to delete in the data sources pane.
2. Click the **Remove Relationship** button.
The relationship is removed from the XML Publisher.

Adding Data to a Report

Once you have added one or more data sources to the data sources panel, you can specify the data you want to include in your XML report. This section describes how to add data to a report and how it is represented on the XML Publisher canvas.

This section covers the following topics:

- [“How to Add Data to a Report”](#) on page 959
- [“Example: Dropping a Repeating Node”](#) on page 960
- [“How Data is Represented on the Canvas”](#) on page 961
- [“More About the Navigation Bar”](#) on page 963

How to Add Data to a Report

There are essentially two ways to add data to a report:

- *Automatically.* You can drag a node from the data sources panel and drop it on the canvas. When you do, Stylus Studio displays a short-cut menu that displays the types of components you can create based on the node you have selected. For example, if you select a repeating element, you can create components that loop – tables and lists, for example.
- *Manually.* You can add an empty component to the report using the main menu (**Report > Insert List**, for example) or the canvas short-cut menu (right-click on the canvas), and then populate the component by dragging and dropping nodes from the data sources panel. Alternatively, you can specify Context and XPath properties in the **Properties** window for the component you want to populate.

The benefit of using the automatic method is that Stylus Studio determines the context and XPath settings required to return the data you have selected. In addition, when you drop

the node on the canvas, Stylus Studio displays on the short-cut menu only those choices that are applicable to the node you selected from the data sources panel.

Example: Dropping a Repeating Node

As described earlier, the data source – whether it is an XML document, a relational database table, an EDI file converted to XML, or some other XML data source – is represented as a data model in the data sources panel. The glyphs used for the nodes are based on the object they represent in the data source, as shown here.

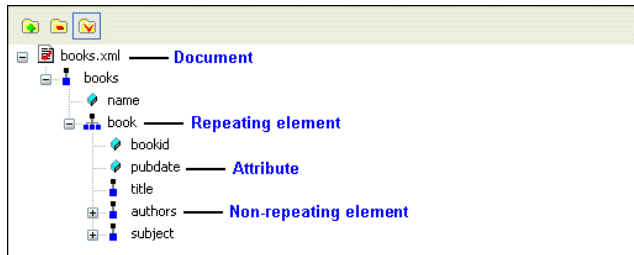


Figure 420. Glyphs Used to Represent a Data Source

For this example, we use `books.xml` as the report's data source. When we drop the `book` repeating element on the canvas, we select **Insert Table > Populated Columns** from the short-cut menu. Stylus Studio creates a table with five columns, one for each of the child nodes in the `book` repeating element, as shown in [Figure 422](#).

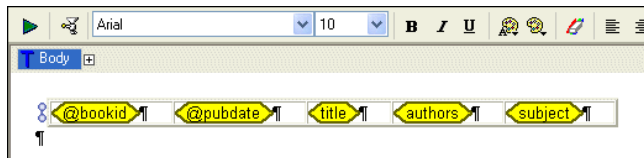




Figure 421. Table Created Automatically Using Repeating Element Child Nodes

The following table summarizes the types of components you can create and automatically populate with data based on the node type.

Table 145. Components for Repeating and Non-Repeating Nodes

<i>Repeating Nodes</i> 	<i>Non-Repeating Nodes</i> 
Text	Text
Image	Image
If	If
Table (either a table with three empty columns, or a table with one column for each child node of the repeating element)	–
Repeater	–
List	–

Note You cannot drag document nodes.

See [“Working with Report Components”](#) on page 966 for information about specific components.

How Data is Represented on the Canvas

Data is represented by glyphs that contain an XPath expression. The composition of these XPath expressions varies based on the context of the component in which the data is being included. The glyph might contain just an element name, or it might display a longer XPath expression if it represents data whose context is not established by the containing component.

Example

The context for the table component shown in [Figure 422](#) is the `video` repeating element from the `videos.xml` document. You can see this if you select the table in the canvas and look at either the

- The Context and XPath sub-properties for the repeating row

- The **Video** table glyph in the navigation bar

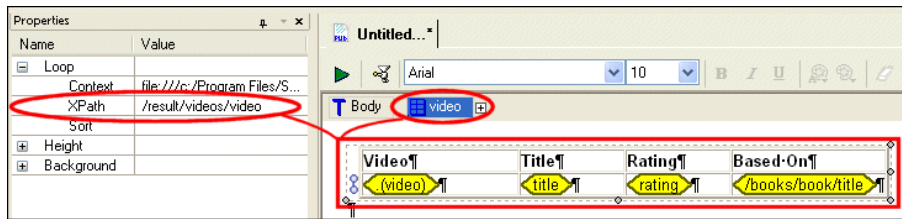


Figure 422. Value Glyphs on the XML Publisher Canvas

Tip You can see the context for any piece of data or component by hovering the mouse pointer over it. When you do, Stylus Studio displays a tooltip that includes the URL and XPath.

The following table shows the different types of XPath expressions you might see in an XML Publisher report.

Table 146. Explanation of XPath Expressions in Data Glyphs

<i>Data Glyph</i>	<i>Contains</i>	<i>Description</i>
	The current context	The XPath expression for context is a dot. To make this easier to see in the glyph, Stylus Studio adds the element name in parentheses following the dot.
	An element (or attribute) name	If the context is established by the containing component, the data glyph contains only the element or attribute name.
	A full XPath	If the context for the data is not established by the containing component, Stylus Studio displays the full XPath needed to resolve it.

More About the Navigation Bar

As you add components to your report, Stylus Studio adds glyphs that represent them to the canvas navigation bar. You can click these glyphs to place the editor's focus on a specific component; similarly, when you select a component from the canvas, the glyphs in the navigation bar change to reflect the editor's current focus.

Consider the following report – it contains two tables, each with a number of columns, and some text headings.

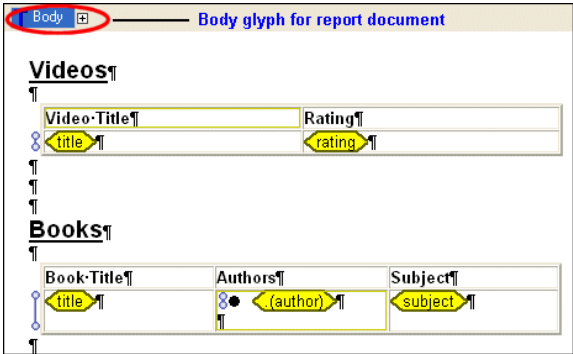


Figure 423. Report Body Glyph Collapsed by Default

The **Body** glyph in the navigation bar represents the report's body. The dark blue means that the report body – the table headings and empty paragraph markers – has the editor's focus. In other words, any editing performed now – changing the text to italic, or making the background a different color, for example – would affect every object in the report body.

The plus sign next to the **Body** glyph indicates that the report body has at least one child. Our report has two children – the table containing video data, and the table containing book data. If we click the plus sign, Stylus Studio displays a drop-down menu that lists the children of the report body, and we see the entries for the video and book tables.

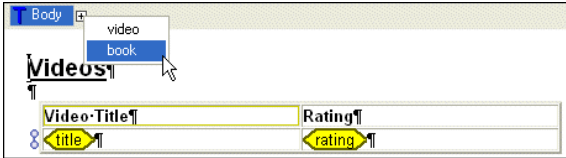


Figure 424. Plus Sign Indicates Children

Click the Glyph to Navigate

You can use the glyphs in the navigation bar to quickly change the editor's focus to the component you select. If we select book from the **Body** glyph drop-down menu, for example:

- The editor's focus moves to the book table. Notice the dashed line around the table in [Figure 425](#).
- The navigation bar changes to reflect the editor's focus.

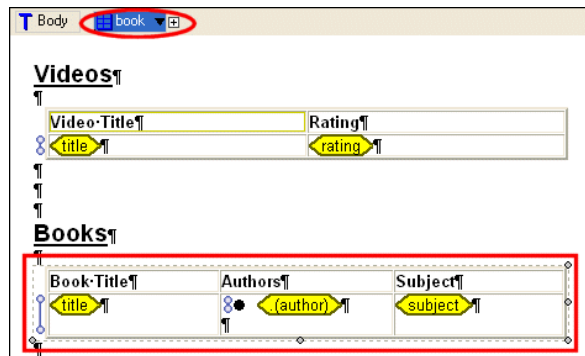


Figure 425. Clicking Navigation Bar Glyphs Changes Editor Focus

Notice that the book glyph, which represents the table containing book data, has two symbols to its right:

- The plus sign, which indicates that the table has children. A table's children are its cells.
- The down arrow, which indicates that the table has siblings. In this example, the table containing video data is the sibling of the current table.

If we now click a cell directly, say, the cell containing author data, notice how the navigation bar changes:

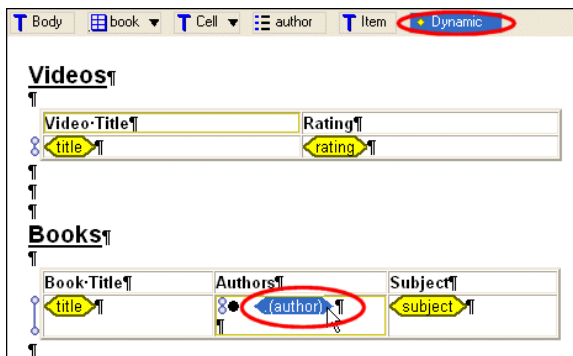

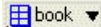

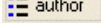

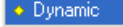


Figure 426. The Navigation Bar Operates as a Tree Based on Report Context

In this fashion, the navigation bar operates a tree, always showing you the report component that currently has focus. Components are displayed from the most general to the most specific. Look at the navigation bar in Figure 426. When the author glyph in the table is selected, the glyphs in the navigation bar are interpreted this way (from left to right):

Table 147. Explanation of Example Navigation Bar Tree

Navigation Bar Glyph	Represents
 Body	The entire report. Every other component is a child of the body component.
 book ▼	The book table. The down arrow means that the table has siblings.
 Cell ▼	The cells in the book table. The down arrow means that the cell has siblings – the other cells in the table.
 author	The list component built on the author element.
 Item	An item in the list component.
 Dynamic	The dynamic value of the author element. It is dark blue because in our example (Figure 426), it is the report component that currently has focus.

Notice the video table is not represented in the navigation bar. This is because the current context in the report canvas is owned by the book table. You can quickly change context in the canvas by either

- Clicking a cell in the video table or the video table itself
- Using the down arrow on the table glyph (which currently displays book) to select video

Note Repeater and text block components have their own glyphs.

Working with Report Components

This section describes the types of components you can include in an XML Publisher report and how to create and work with them.

This section covers the following topics:

- [“Types of Components”](#) on page 966
- [“Tables”](#) on page 967
- [“Lists”](#) on page 970
- [“Text”](#) on page 972
- [“Images”](#) on page 973
- [“Repeaters”](#) on page 976
- [“Ifs”](#) on page 977
- [“Component Properties”](#) on page 980
- [“Formatting Components”](#) on page 984

Types of Components

There are two types of components you can include in an XML Publisher report:

- *Visual components*; these are components that have a visual representation in the published HTML or XSL-FO report. Examples include tables and lists.
- *Non-visual components*; these are components that do not have a visual representation in a finished report. Examples include repeaters and ifs.

All components, regardless of type, are represented graphically on the XML Publisher canvas.

The following table lists all XML Publisher report components and tells you where to find more information about them.

Table 148. XML Publisher Report Components

Component Name	Component Type	For More Information
Table	Visual	See “ Tables ” on page 967
List	Visual	See “ Lists ” on page 970
Text	Visual	See “ Text ” on page 972
Image	Visual	See “ Images ” on page 973
Repeater	Non-visual	See “ Repeaters ” on page 976
If	Non-visual	See “ Ifs ” on page 977

Tables

A *table* is a visual component that usually iterates over the data it contains. You typically create a table when you want your report to display multiple rows with two or more columns of dynamic data – one row for each movie in the `videos.xml` file, with each row containing the movie’s title, its genre, and its rating, for example.

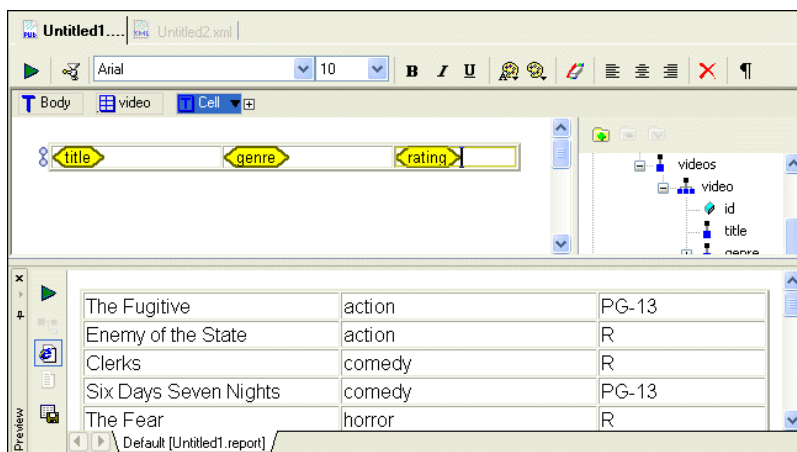


Figure 427. Example Table and Output

If your data can be displayed in a single column (only movie titles, for example), you might want to consider using either the list or repeater components. See “Lists” on page 970 and “Repeaters” on page 976 for more information.

Creating a Table

The easiest way to create a table in XML Publisher is to drag and drop a repeating element. When you drop the repeating element on the canvas, Stylus Studio displays a short-cut menu with an **Insert Table** choice. You can insert a table with either

- **Populated Columns** – Stylus Studio creates a table with one column for each of the child nodes of the repeating element. The table’s context and XPath are set based on the repeating element used to create it. Each column contains a data value glyph representing a child node.
- **Empty** – Stylus Studio creates table with three empty columns. As with the previous option, the table’s context and XPath are set based on the repeating element used to create it, but it is up to you to select from the data sources pane the child nodes you want to include in the table.

You can also create a table manually (**Report > Insert > Table**, or select **Insert Table** from the canvas short-cut menu). When you create a table like this, however, the context and XPath are not set for you, and it will only contain the number of rows you explicitly create for it unless you also define the Loop property for a row.

Graphical Representation

Tables, like the one shown in [Figure 428](#), are displayed as a single row with a loop symbol (⌘); the loop symbol indicates a repeating row.

[Figure 428](#) shows a table based on the book repeating element in `books.xml` that was created using the **Populated Columns** short-cut menu choice.



Figure 428. Table Created with Populated Columns

[Figure 429](#) shows a table based on the same repeating element, but it was created using the **Default Columns** short-cut menu choice.



Figure 429. Table Created with Default Columns

Finally, [Figure 430](#) shows a table that was created manually using the **Report** menu. Notice that it does not have the loop symbol associated with repeating rows.



Figure 430. Manually Created Table

Sorting

By default, data for dynamic rows is displayed in document order. You can use an XPath expression in the Loop property's Sort sub-property to specify a different sort order. The Loop property appears on the **Row** tab of the **Properties** window.


Adding Rows and Columns

◆ To add rows and columns to a table:

1. Select the cell before or after which you wish to add a row or column.
2. Right-click.
Stylus Studio displays a short-cut menu.
Alternative: Click the **Report > Table** menu.
3. Select the appropriate choice from the menu.

Deleting Rows, Columns, and Tables

◆ To delete a row, column, or table:

1. Select a cell in the row or column you want to delete.
2. Click the **Delete** button in the toolbar ()
Stylus Studio displays a drop-down menu.
Alternative: Right-click.
Alternative: Click the **Report > Table** menu.
3. Select the appropriate choice from the menu.

Lists

A *list* is visual component that iterates over the data it contains and contains one or more items. You typically create a list when you want your report to display a list of dynamic values – all the books, by title, in the `books.xml` document, for example.

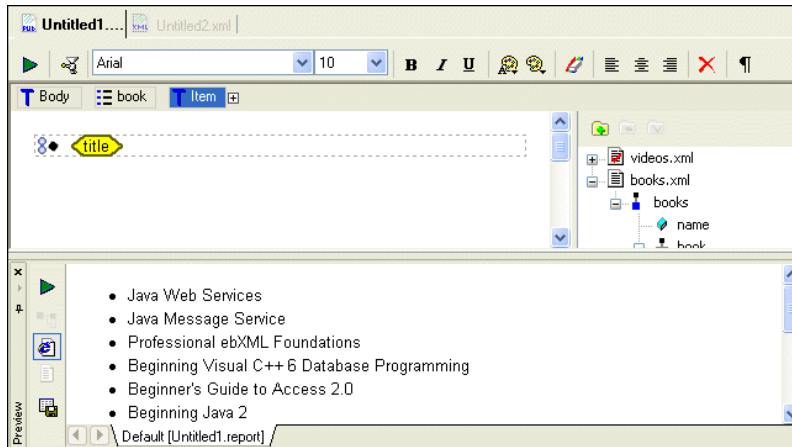


Figure 431. Example List and Output

Lists are formatted using bullets, but you can choose numeric, alphabetic, and other symbols like squares and circles. Depending on your needs, you might prefer to use the repeater or table components for dynamic data. See [“Repeaters”](#) on page 976 and [“Tables”](#) on page 967 for more information.

Creating a List

The easiest way to create a list in XML Publisher is to drag and drop a repeating element. When you drop the repeating element on the canvas, Stylus Studio displays a short-cut menu with an **Insert List** choice. The list’s context and XPath are set based on the repeating element used to create it. You specify the data you want the list to contain by dragging the appropriate node from the data sources panel and dropping it in the list as Value.

You can also create a list manually (**Report > Insert > List**, or select **Insert List** from the canvas short-cut menu). When you create a list like this, however, the context and XPath are not set for you, and it will iterate over the data you specify only if you also define the Loop context and XPath properties for an item.

Graphical Representation

As shown in [Figure 431](#), a list is represented as a bounding box drawn with a dashed line containing a bullet symbol and, usually, a loop symbol (8). If you created the list manually, the loop symbol appears only if you specify the Loop property's Context and XPath sub-properties for an item.

Sorting

By default, data for dynamic lists is displayed in document order. You can use an XPath expression in the Loop property's Sort sub-property to specify a different sort order.


Adding Items

◆ To add items to a list:

1. Select the item before or after which you wish to add a new item.
2. Right-click.
Stylus Studio displays a short-cut menu.
Alternative: Click the **Report > List** menu.
3. Select the appropriate choice from the menu.

Deleting an Item or a List

◆ To delete an item or a list:

1. Select the item or list you want to delete.
2. Click the **Delete** button in the toolbar ()
Stylus Studio displays a drop-down menu.
Alternative: Right-click.
Alternative: Click the **Report > List** menu.
3. Select the appropriate choice from the menu.

Text

A *text* component is a block that allows you to create an area for text that can be formatted independently from the body text in a report. The text component in Figure 432 has been formatted using a crimson italic font, which differs from the default body text that precedes and follows it.

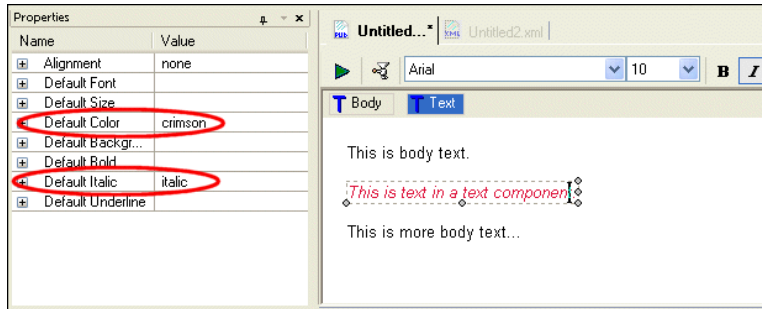


Figure 432. Body Text and Text Component

Text components and body text have the same properties (Alignment, Font, Color, Size, and so on).

Creating a Text Component

◆ **To create a text component:**

1. Click the canvas where you want to insert the text component.
2. Select **Report > Insert > Text** from the menu.

Alternative: Right-click and select **Insert Text** from the short-cut menu.

Graphical Representation

In the XML Publisher canvas, a text component is represented, when selected, as a bounding box drawn with a dashed line. If the text component is not selected, the bounding box does not appear.

Images

An *image* is a component that contains a GIF, JPEG, or some other graphic file. You can place image components within other components (like tables, lists, and repeaters, for example), or directly on the report body.

Creating an Image

◆ **To create an image component:**

1. Click the canvas where you want to insert the image component.
2. Select **Report > Insert > Image** from the menu.
Alternative: Right-click and select **Insert image** from the short-cut menu.
3. Specify the location of the image file(s). See “[Specifying an Image Source](#)” on page 974 for more information.

Note

The location of all image files must be specified relative to the target destination of the HTML+CSS or XSL-FO. For example, do not specify `c:\myFiles\images` as a source directory unless that directory is accessible to the finished report.

Graphical Representation

In the XML Publisher canvas, an image is represented as a small square with a cross-hatched pattern. [Figure 433](#) shows an image component that includes some text to its right – (*photo of the author*).

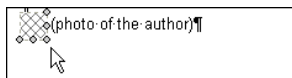


Figure 433. Image (with text)

Images are resolved when you preview the report only if the image files are accessible to Stylus Studio using the source information you have specified. Unresolved images are rendered as red Xs, as shown in [Figure 434](#):



Figure 434. Symbol for Unresolved Image in Preview Window

Specifying an Image Source

You can use static and dynamic images in a report. A *static* image is one that never changes. An example of a static image is a corporate logo that appears in a fixed place on a report. To specify a static image, just the complete file URL in the image's Source property – `file://c:\MyProjects\Images\StylusLogo.gif`, as shown in [Figure 435](#).

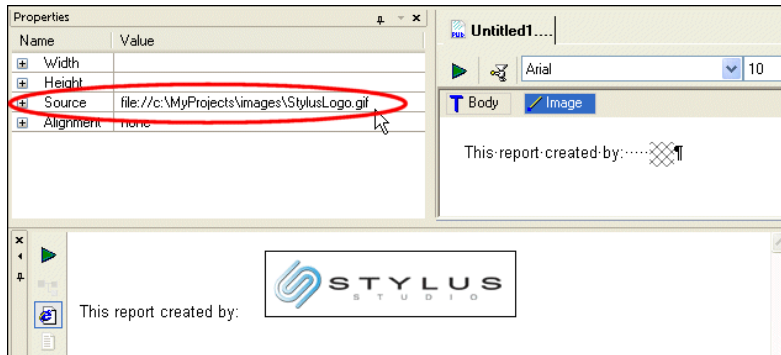


Figure 435. Example of a Static Image

A *dynamic* image is one whose source changes based on the context defined for it. An example of a dynamic image is the cover art for the movies in the `videos.xml` document – the image varies based on the video id as determined by the current context.

To specify a dynamic image, you need to define the image component's Source property's Context and XPath sub-properties:

- Context – defines the document context for the evaluation of the Source property's XPath sub-property. This can be a source document URL, or a variable.
- XPath – an XPath expression used to evaluate the source document; the context for the XPath expression is determined by the Context sub-property. This can be any XPath expression.

When you specify these properties, Stylus Studio displays `<dynamic>` in the Source property field.

Note Regardless of whether you are using static or dynamic images, the image source must be available to the finished HTML+CSS or XSL-FO report.

Example: Specifying a Dynamic Image Source

The cover art for the movies in the `videos.xml` document are written to the `\examples\VideoCenter\images\video\` directory where you installed Stylus Studio. The

name of each .gif file is the same as the value of the id attribute of the video repeating element. We want to create a simple table displaying the movie's title and its cover art, as shown in [Figure 436](#).

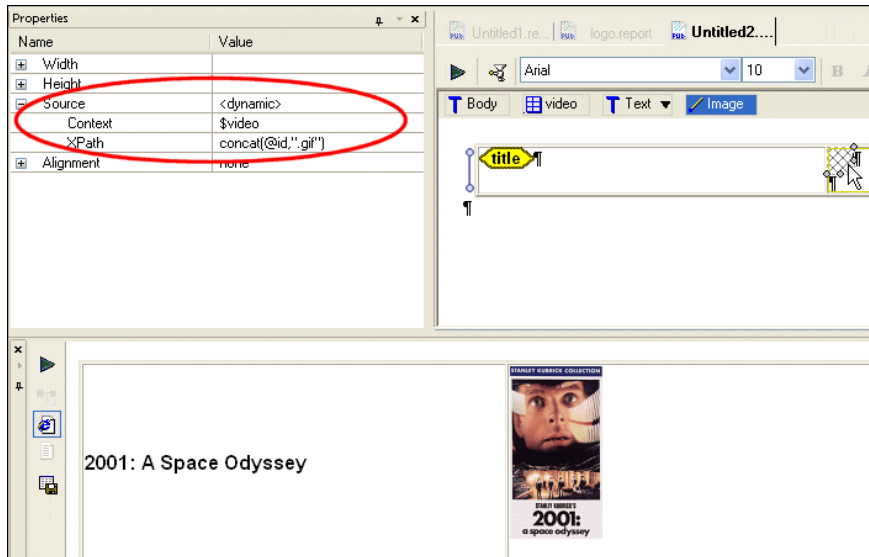


Figure 436. Example of Dynamic Images

To create this report in XML Publisher, we would:

1. Create a table by dragging the `video` repeating element and dropping it on the canvas. This establishes the context for the cells in the table, as well as creating a value, `$video`, that represent this context.
2. Drag and drop the `title` element in the table's first cell (as a value).
3. Right-click and insert an image component in the table's second cell.
4. Specify the context for the image:
 - a. Change the image component's Context sub-property to `$video`. The context for this variable (`result/videos/video`) was established automatically when we used the `video` repeating element to create the table.
 - b. Change the image component's XPath sub-property to `concat(@id)\".gif\"`. This concatenates the value of the current `id` attribute with the string `.gif` to identify image file to display. The source for these image files is specified in the following step.

5. Specify the source for the image files – change the body component’s Base URI property to `c:\Program Files\Stylus Studio\examples\VideoCenter\images\video\`, or wherever you have installed Stylus Studio.

Specifying Image Size

By default, Stylus Studio displays the image in the finished report using the source file’s dimensions. You can use the image component’s Width and Height properties to specify a different size. When you do this, Stylus Studio changes the dimensions of the image glyph (see [Figure 433](#)) on the canvas to reflect the change.

Repeaters

A *repeater* is a component that iterates over data in a data source based on the context defined for it. When the report is executed, a new line is added to the report for each new value. These lines are not formatted in any way, so, depending on your needs, you might prefer to use the list or table components for repeating data. See “[Lists](#)” on page 970 and “[Tables](#)” on page 967 for more information.

Creating a Repeater

The easiest way to create a repeater in XML Publisher is to drag and drop a repeating element. When you drop the repeating element on the canvas, Stylus Studio displays a short-cut menu with an **Insert Repeater** choice. The repeater’s context and XPath are set based on the repeating element used to create it. You specify the data you want the repeater to contain by dragging the appropriate node from the data sources panel and dropping it in the repeater as Value.

You can also create a repeater manually (**Report > Insert > Repeater**, or select **Insert Repeater** from the canvas short-cut menu). When you create a repeater like this, however, the context and XPath are not set for you, and it will iterate over the data you specify only if you also define the Loop context and XPath properties for an item.

Graphical Representation

In the XML Publisher canvas, a repeater is represented as a bounding box drawn with a dashed line, usually with a loop symbol (∞) on the left side of the bounding box. [Figure 437](#) shows a repeater based on the `title` element in `videos.xml`. It was created by

1. Dragging the `video` repeating element on the canvas and selecting **Insert Repeater** from the short-cut menu.

This action defines the context for the iterative action.

2. Dragging the `title` element and dropping it inside the repeater.

This action defines the data to be included in the repeater.



Figure 437. Repeater with a Text Value

The loop symbol is present only if the repeater's Loop property has values specified for its Context and XPath sub-properties. You can specify these properties manually in the **Properties** window, but it is usually easier to create the repeater by dragging a repeating element, dropping it on the canvas, and choosing **Insert Repeater**.

Sorting

By default, data for the repeater component is displayed in document order. You can use an XPath expression in the Loop property's Sort sub-property to specify a different sort order.

ifs

An *if* is a component that represents a condition (if... then... else...). You can use if components to control report content. You can insert if components within other components (within a table cell, for example).

Creating an If

The easiest way to create an if component in XML Publisher is to select **Report > Insert > If** from the Stylus Studio menu, or select **Insert If** from the canvas short-cut menu.

The if component's context is established automatically only if you insert within another component – such as a table or list – whose context is already set. The if component does not inherit its context from the body component.

Graphical Representation

In the XML Publisher canvas, an if component is represented as two tabs, **true** and **false**, within a bounding box drawn with a dashed line. [Figure 438](#) shows an empty if component.



Figure 438. If Component

Example

We need to create a report that contains a simple table that lists the title and rating for all movies in the `videos.xml` document. In addition, if the movie carries an ‘R’ rating, we want to display the R-rated symbol (**R**) for emphasis. A sample of the report is

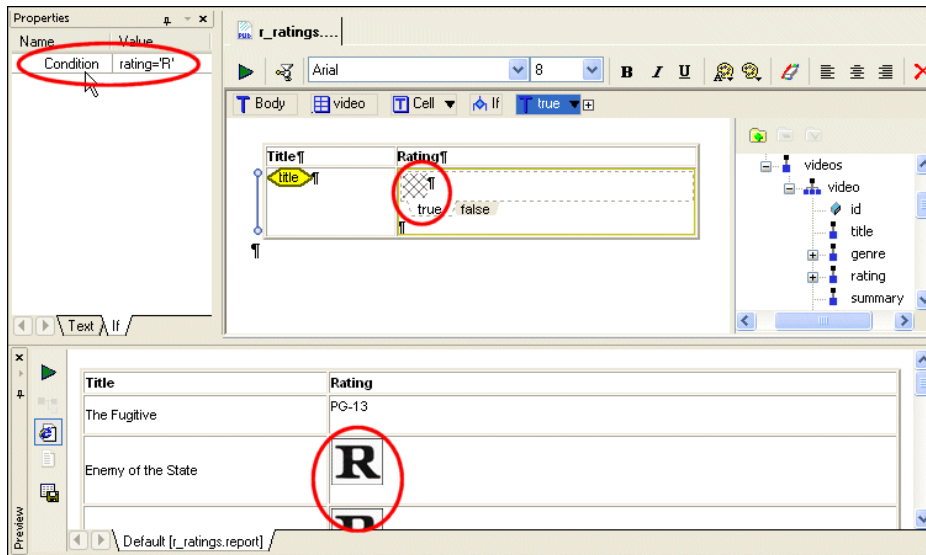


Figure 439. Example of an If Component

To create this report in XML Publisher, we would:

1. Create a table by dragging the `video` repeating element and dropping it on the canvas. This establishes the context for the cells in the table.
2. Drag and drop the `title` element in the table’s first cell (as a value).

3. Right-click and insert an if component in the table's second cell.
4. Select the if component and click the **If** tab in the **Properties** window.
5. Set the Condition property to `rating='R'`.
6. Specify the true condition:
 - a. Select the **true** tab.
 - b. Right-click and select **Insert Image**.
 - c. Set the image component's Source property to the location of the image we want to display for R-rated movies (`c:\MyProjects\images\r_rating.gif`, for example).

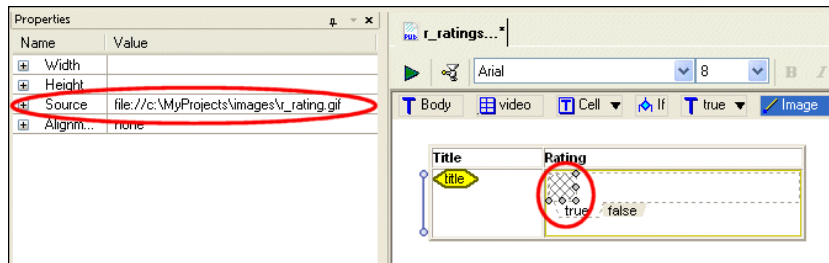


Figure 440. Setting the True Condition

7. Specify the false condition:
 - a. Select the **false** tab.
 - b. Drag the rating node from the data sources panel and insert it as a value.

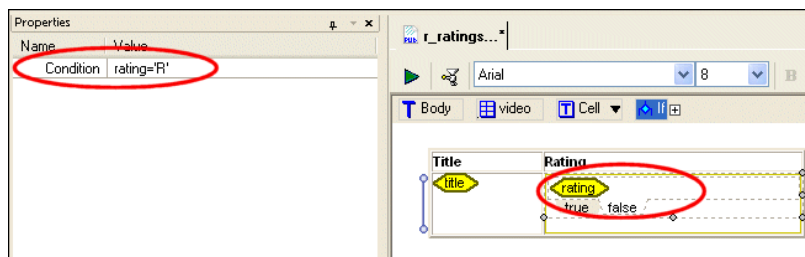


Figure 441. Setting the False Condition

Component Properties

Each component in a report – the body, tables, lists, repeaters, and so on – is associated with a set of properties that control its formatting and content. The text component has properties for **Color**, **Font**, **Size**, and so on. The **Color** property, for example, lets you select *aqua*, *bisque*, *blue*, and so on. Properties vary based on the component. (See “[Properties Reference](#)” on page 999 for a complete list.)

Context and XPath Sub-Properties

Each property has **Context** and **XPath** sub-properties that let you define the conditions under which you want to, say, display a value or apply a given formatting characteristic. You can use the **Color** property’s **Context** and **XPath** sub-properties to format text based on the value returned by an XPath expression – format all ‘R’ rated movies in `videos.xml` using the color red, for example. (See “[Example: Using Context and XPath Sub-Properties to Format Text](#)” on page 981 later in this section for an illustration of this technique.)

The Properties Window

Component properties are displayed in the **Properties** window, a dockable window you can place anywhere on your desktop.

- ◆ To display the Properties window, click **View > Properties**

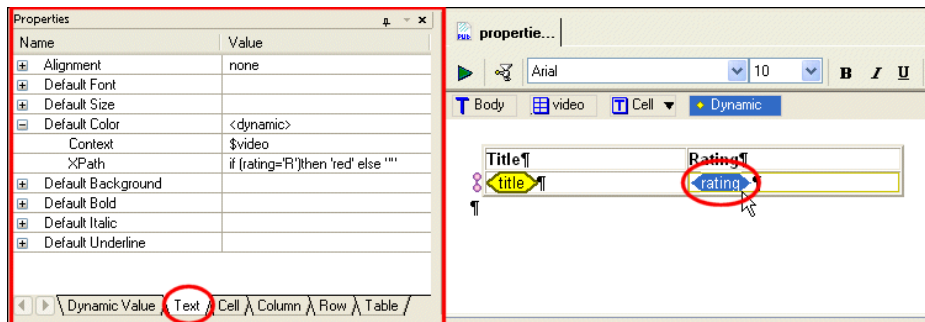


Figure 442. Properties Window

The **Properties** window consists of one or more tabbed pages. The specific tabs that are present in the **Properties** window vary based on the component you have selected in the report canvas. As shown in [Figure 442](#), the **Properties** window for the rating element in the video table includes tabs for the

- The context and XPath expression that return the dynamic value of the rating element
- Text value
- Cell
- Column
- Row
- Table

The order of the tabs in the **Properties** window reflects the hierarchy of the currently selected component, from the most specific (the currently selected component) to the most general (the parent component to which it belongs), left to right. The report body is the only component without tabbed pages.

Example: Using Context and XPath Sub-Properties to Format Text

Our report contains a table based on the `video` element from `videos.xml`; the table lists the movie title and rating. When we preview the report, we see that if a movie's rating is 'R' the rating is displayed in red, while other movie ratings are displayed using the default color.

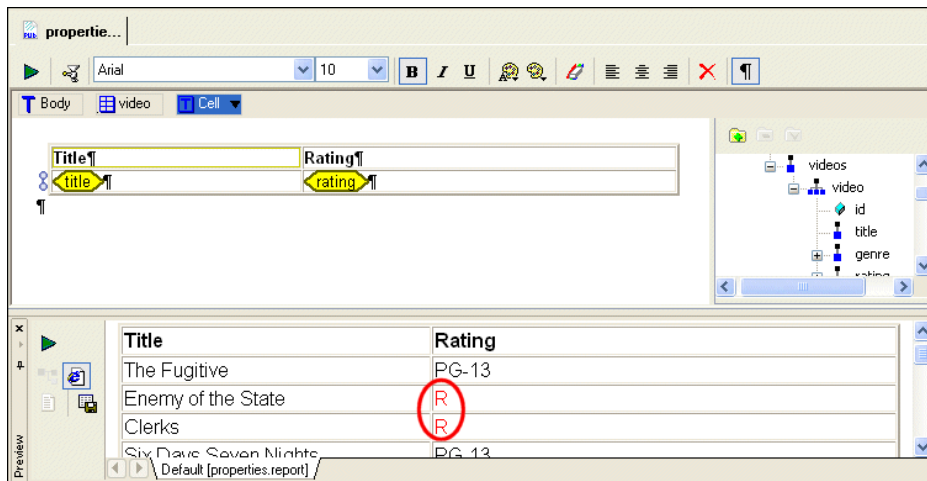


Figure 443. Preview of an XML Publisher Report

If we look at the properties for the rating element, we can see how this was achieved.

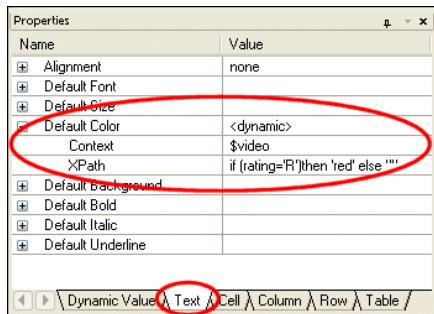


Figure 444. Properties for the Rating Element

As you can see, **Default Color** property is specified as `<dynamic>`. Stylus Studio sets the **Default Color** property to this value automatically when you specify

- A context for the evaluation of an XPath expression defined in the **XPath** property. Here, the context is `$video`. Stylus Studio created this variable, which represents the `videos.xml` document, when we created the table using the `video` repeating element from this document.
- An XPath expression to be evaluated in the context defined by the **Context** property. Here, the XPath expression is `if (rating = 'R') then "red" else ""`.

Together, these properties control when data from the `rating` element in the `videos.xml` document is set to the color red.

Entering XPath Expressions

You can enter an XPath expression by typing directly in the **XPath** field. If you prefer, you can use the **XPath Editor** dialog box, shown in [Figure 445](#).

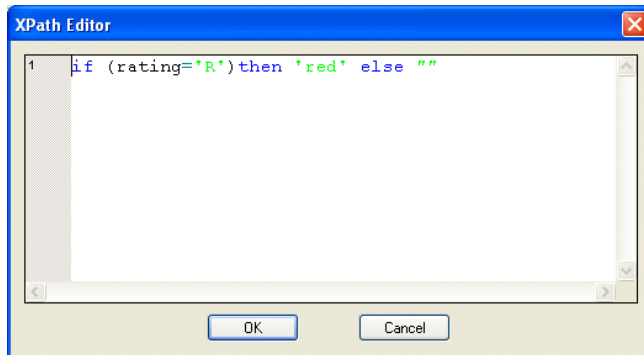


Figure 445. XPath Editor Dialog Box

The **XPath Editor** dialog box supports Stylus Studio's Sense:X auto-completion and text coloring, which can provide useful prompts as you type your XPath expression.

◆ **To display the XPath Editor dialog box:**

1. Click the **XPath** entry field for the property you want to define.
The text cursor and a “more” button appear in the field.

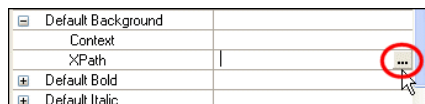


Figure 446. Opening the XPath Editor Dialog Box

2. Click the “more” button.
Stylus Studio displays the **XPath Editor** dialog box.

Formatting Components

Formatting in XML Publisher works in a fashion similar to formatting in many text and graphical editors – you select the item you want to format and then apply a format from a tool bar, menu, or palette. An *item* can be

- One or more words
- A component like a list or table

As described later in this section, the effect of applying a format varies based on where and how you apply it.

Formats

Formats include font, background and foreground (that is, text) color, size, and alignment. For a complete list of available formats, see “[Text Properties](#)” on page 1003.

Ways to Apply Formats

There are several ways to apply formats to XML Publisher reports. You can use

- The tool bar, which is located above the XML Publisher canvas

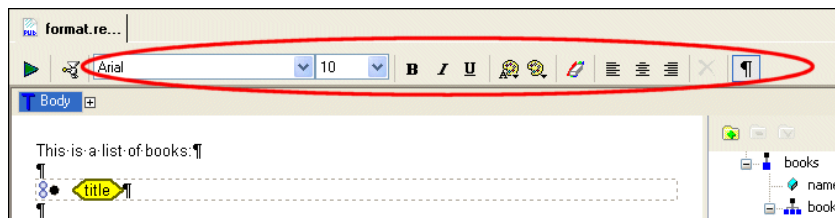


Figure 447. XML Publisher Formatting Tool Bar

- The **Report** and short-cut menus

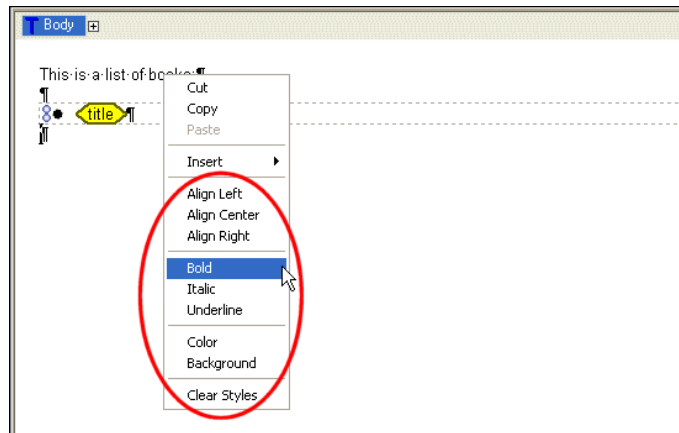


Figure 448. Format Choices on Short-Cut Menu

- The **Properties** window

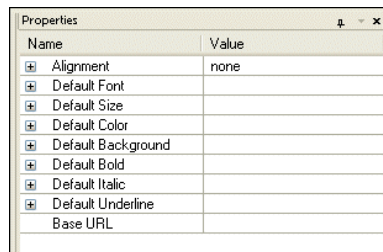


Figure 449. Format Properties

Each of these methods behaves in a similar fashion in that they apply a format to the currently selected item. The **Properties** window, however, is slightly different – in addition to applying a format, it sets the default for the component that currently has focus. See “[Setting Default Properties](#)” on page 988 for more information.

Formatting Influenced by Component Hierarchy

Since all components in a report occur within the context of the body component, any formatting you perform to the body component affects every component it contains – every table, list, text component, and so on – unless that component has a default value for the same format property specified on the **Properties** window. Put another way, any

formatting applied to a parent component affects all of its children unless that child has a default value specified for it.

Consider the following example report, which shows an introductory sentence (body text) and a list of book titles (list component).

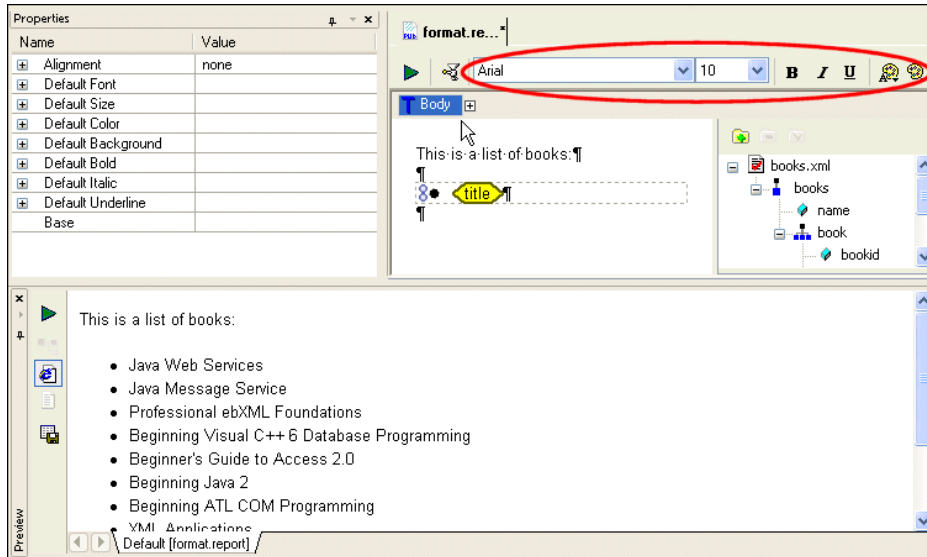


Figure 450. Default Format Settings

All of the report's components are displayed using the XML Publisher default settings for the body component, as seen in the toolbar (font is Arial, size is 10, non-bold, non-italic, no underline, and so on).

If we now click the **Bold** button on the toolbar, all of the report's text, including the list component items, is rendered in bold – the list component is a child of the body component, so all formatting done to the body cascades to the list as well.

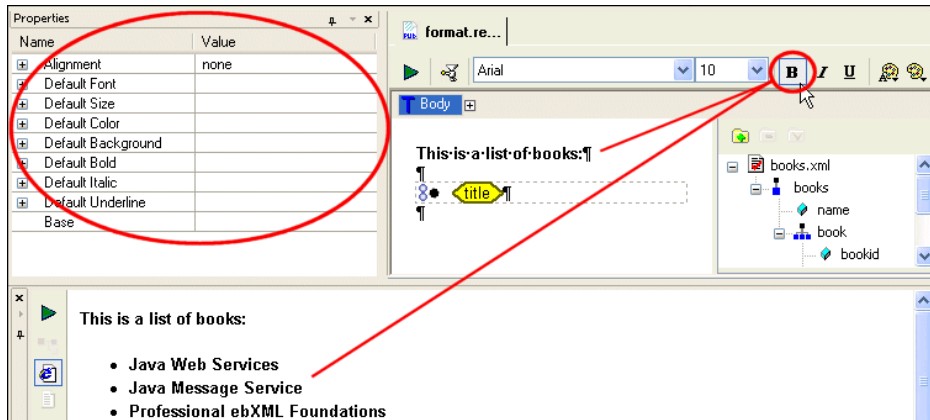


Figure 451. Using the Bold Tool to Format the Entire Report

Notice that the values in the **Properties** window have not changed – although we have changed some of the formatting characteristics for the body component, we have not set any of its default values.

Next, we select just the `title` value glyph and click the **Italic** button.

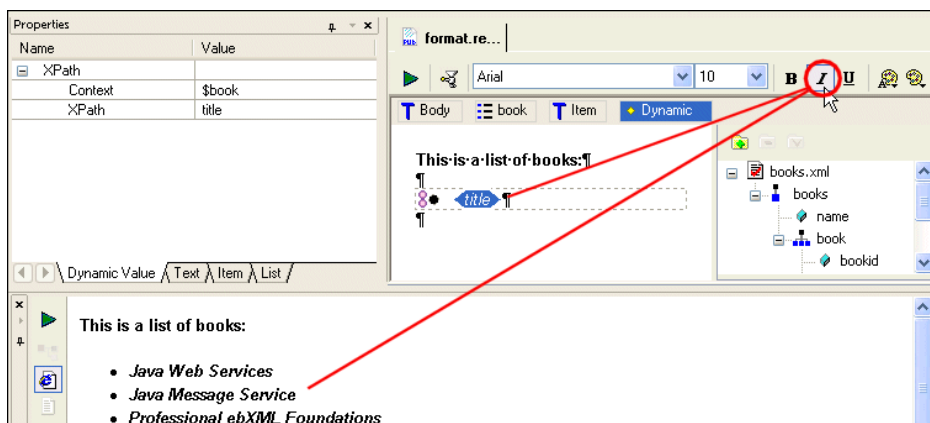


Figure 452. Using the Italic Tool to Format Only Data Values

Notice that the label in the value glyph has been italicized. And when we preview the report, we see that the text representing each data value (each book's `title` element)

- Retains the formatting established for the body (bold)
- Includes the formatting specified just for it (italic)

The value glyph has a format state separate from the body, and this state is reflected in both the glyph (the label is italicized) and the tool bar (when the glyph is selected, the italic tool in the tool bar is highlighted).

Setting Default Properties

Default properties allow you to specify formatting for a child component that differs from that of its parent components. These settings remain in effect regardless of how the formatting for its parents changes. Return to the example illustrated in [Figure 452](#) – all of the report's text is bold (we formatted the body component using the bold tool), and the list text is also italic (we formatted the title value glyph).

If we do not want the list text to appear in bold, we need to establish a default for it – otherwise, it will continue to inherit the bold setting from its parent, the body component. So, as shown in [Figure 453](#), we click the glyph to give it focus, and then change the **Default Bold** setting to **Normal**.

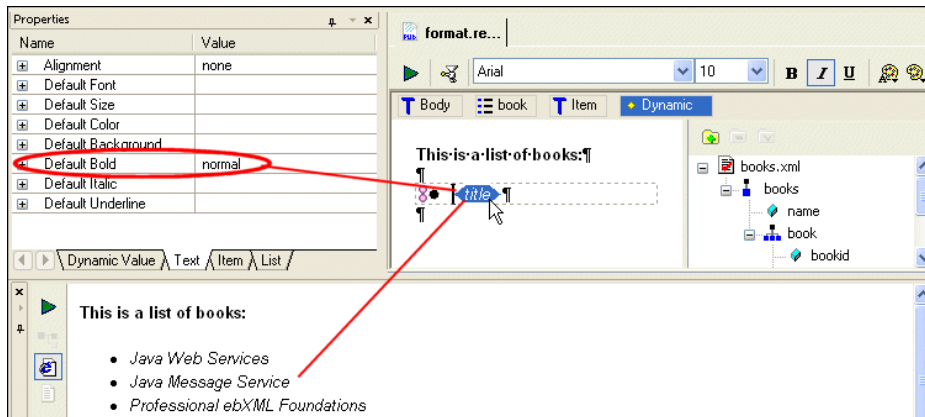



Figure 453. Default Settings Take Precedence


Clearing Formats

You can clear formats you have applied using the **Clear Styles** button () on the tool bar or the **Report (Report > Text > Clear Styles)** or short-cut menus. Format properties are removed (or cleared) in the same way that they were applied – that is, on a component-by-component basis. For example, if the body component text was italic, when you

cleared the formatting the body component text would be returned to its original state. Clear removes all formatting, regardless of whether or not the XML Publisher report was last saved.

Note Clearing formats does not affect a component's default property settings.

◆ **To clear formatting:**

1. Select the report component whose formatting you want to remove.
2. Click the **Clear Styles** button ().

Alternatives: Select **Report > Text > Clear Styles** from the menu, or select **Clear Styles** from the short-cut menu (right-click).

The formatting is removed from the component you selected. Default settings are not affected.

Generating Code for an XML Publisher Report

Once you have built and previewed your report in XML Publisher, you can generate XSLT or XQuery code for it. The generated code includes all the instructions necessary to create the report you composed in XML Publisher.

Tip When you preview the XSLT or XQuery code generated by XML Publisher, the result displayed in the **Preview** window should look the same as the preview of the XML Publisher report.

Supported Languages and Documentation Types

The code generator for XML Publisher supports the following transformation languages:

- XSLT 1.0
- XSLT 2.0
- XQuery 1.0

You can use these languages to generate reports of the following type:

- HTML+CSS
- XSLT-FO

You select these settings, as well as a target for the output file, on the **Generate Transformation** dialog box:

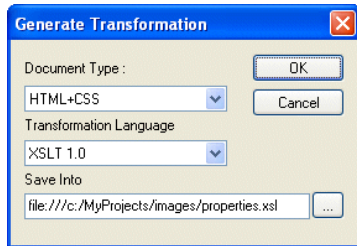


Figure 454. Generate Transformation Dialog Box

Sources

As described in [“Working with Data Sources”](#) on page 945, if you plan on generating XSLT for your XML Publisher report you must have specified at least one default source. (Sources are not required for generating XQuery, though it is considered good practice to specify one.)

Additional Sources

You can specify multiple sources for an XML Publisher report. The first source is specified in the XSLT/XQuery **Scenario Properties** dialog box as shown in the following table.

Table 149. How Additional Source Documents Are Referenced

<i>Language Type</i>	<i>Scenario Property</i>	<i>Referenced in Generated Code As</i>
XSLT	Source XML URL	A global parameter: <code>xsl:param name="input1"</code>
XQuery	Main Input	An external variable: <code>declare variable \$input1 as document-node() external</code>

The expression associated with these variable names is displayed on the **Parameter Values** page of the **Scenario Properties** dialog box, like the one shown for XQuery scenarios in [Figure 455](#).

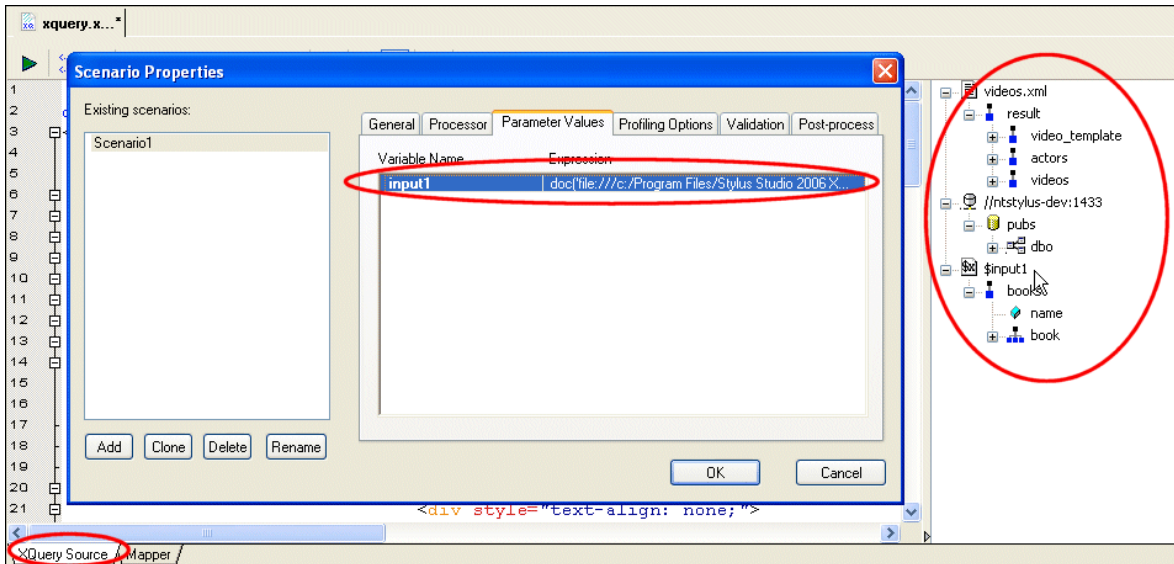


Figure 455. Additional Sources Displayed on XQuery (XSLT) Scenario Properties

As you can also see, [Figure 455](#) shows how data sources specified in XML Publisher are represented in the XQuery (and XSLT) editor. This XQuery uses three XML sources:

- videos.xml (it is the default source)
- A relational source from the pubs data base
- A local copy of books.xml, displayed on the XQuery data sources panel using the variable name, \$input1, with which it is associated

More About Relational Sources

Relational sources, like the one shown in [Figure 455](#), are referenced in different ways by XSLT and XQuery:

- XSLT code references relational data sources within a `document()` function:

```
document('xquery:///jdbc:xquery:sqlserver://ntstylus-dev:1433;table=companies;user=sa;xmlforest=true;schema=dbo;DatabaseName=pubs;urltype=.xml')
```
- XQuery code references relational data sources using the `collection()` function:

```
collection('pubs.dbo.companies')/companies/ticker
```

See “[Working with the XQuery collection\(\) Function](#)” on page 777 for more information.

How to Generate Code

◆ To generate code for an XML Publisher report:

1. Preview the XML Publisher report. If it is satisfactory, continue with this procedure.
2. Click the **Generate** button.

Alternative: Select **Report > Generate** from the menu.

Stylus Studio displays the **Generate Transformation** dialog box.

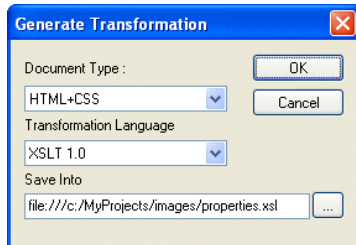


Figure 456. Generate Transformation Dialog Box

3. Use the **Document Type** field to specify whether you want the XQuery or XSLT code you generate to create HTML+CSS or XSL-FO.
4. Use the **Transformation Language** field to specify whether you want to use XSLT 1.0, XSLT 2.0, or XQuery 1.0.
5. The default URI for the generated XSLT or XQuery code is displayed in the **Save Into** field. Unless you specify otherwise, Stylus Studio uses the .report file name for the .xsl or .xquery file name.
6. Click **OK**.

Example: Building an XML Publisher Report

In this example, we will build a simple XML report based on `videos.xml`. This XML document is in the `VideoCenter` folder of the `examples` project.

This section covers the following topics:

- “Getting Started” on page 993
- “Insert and Populate a Table” on page 993
- “Simple Table Formatting” on page 995
- “Format Data Conditionally” on page 996
- “Generate the Code” on page 998

Getting Started

In this part of the procedure, we create a new XML Publisher report and specify a data source.

◆ **To get started:**

1. Select **File > New > XML Report** from the Stylus Studio menu.
Stylus Studio displays the XML Publisher Editor.
2. If the **Project** window is not already open, open it (**View > Project Window**).
3. Drag `videos.xml` from the `VideoCenter` folder to the data sources panel in the XML Publisher Editor.

Insert and Populate a Table

Next, we add a table based on the `video` repeating element and populate its columns.

4. Expand the `videos` node.
5. Drag the `video` repeating element from the data sources panel and drop it on the XML Publisher canvas.
6. Select **Insert Table > Empty** from the short-cut menu.

Stylus Studio creates a three-column table. At this point, the XML Publisher Editor should resemble [Figure 457](#):

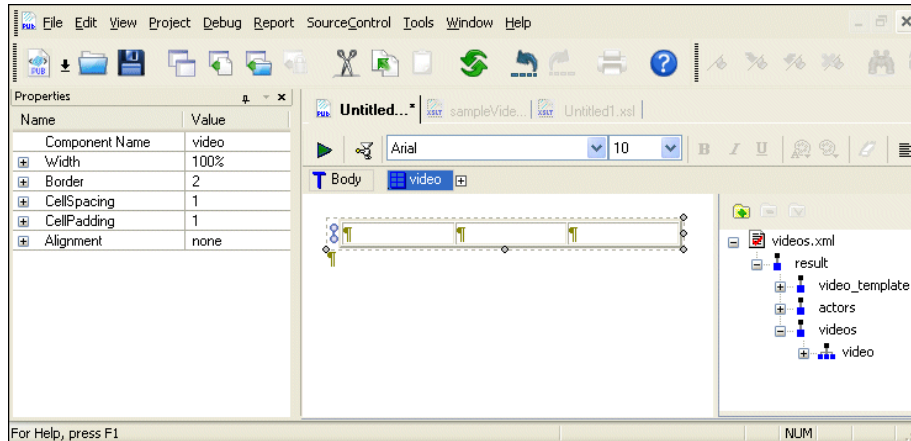


Figure 457. Default Table in New XML Publisher Report

Although the table currently contains no values, if you mouse over the repeating glyph at the left of the table (⌘) you will see that it references the `videos.xml` document and that the current XPath expression evaluates the `video` repeating element (`/result/videos/video`).

7. Right-click any cell in the table and select **Remove Column** from the short-cut menu.
8. Expand the `video` node in the data sources panel.
9. Drag the `title` node and drop it in the first column; select **Insert Value** from the short-cut menu.

A value glyph with the element name appears in the cell.

10. Repeat this step with the `rating` element, dropping it in the second column.
11. Right-click the table again, and select **Add Row Before**.
Stylus Studio adds a new row to the table. Note that the new row does not repeat – the repeating glyph is associated only with the second row, the row that contains the data.
12. Type *Title* in the first column and *Rating* in the second.
13. Click **Preview** (▶).

Stylus Studio creates a three-column table. At this point, your XML Publisher Editor should resemble this:

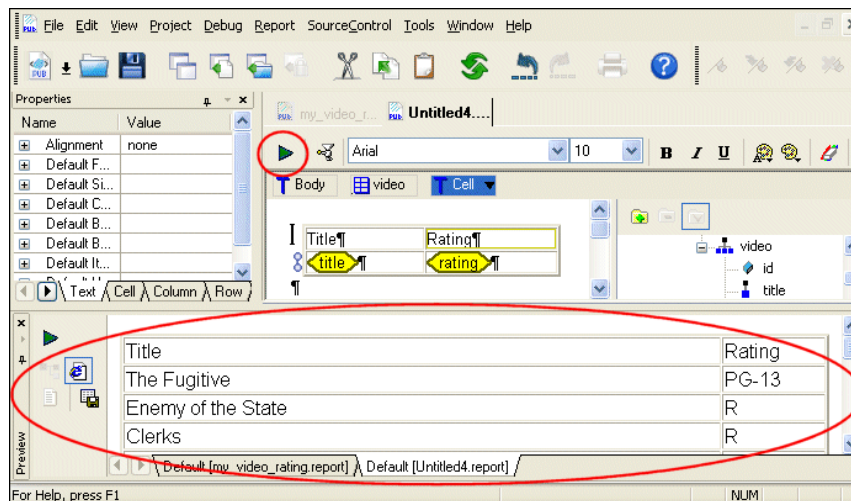




Figure 458. Preview of XML Publisher Report

14. Click the **Hide docked window** button (the small X) to close the **Preview** window. This gives us more room to work, and Stylus Studio will automatically display the **Preview** window the next time we preview the report.

Simple Table Formatting

In this section, we will perform some simple formatting to make the table more presentable. We will start by making the column headings, *Title* and *Rating*, bold.

15. First, click the **Show Text Symbols** button () in the toolbar. This removes text symbols (like spaces and paragraph markers) from the canvas, which can make the canvas easier to work with while you build the report.
16. Click anywhere in the cell containing the *Title* string.
17. Click the **Bold** tool on the toolbar (). The *Title* string appears bold in the canvas.
18. Make the *Rating* string bold.

Next, we want to adjust the width of the first column, so the ratings appear closer to the title.

19. Click either of the rows in the table's first column.
20. Click the **Column** tab in the **Properties** window. (If the **Properties** window is not displayed, select **View > Properties** from the Stylus Studio menu.)
21. Enter 40% for the **Width** property.
22. Click **Preview** (▶).

The **Preview** window displays the changes.

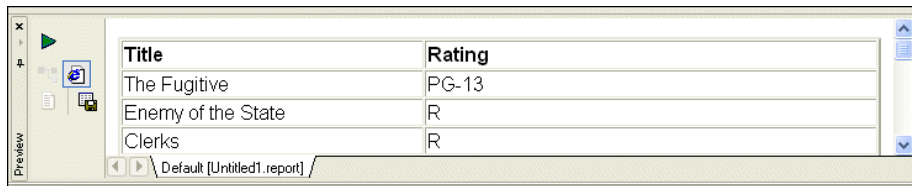


Figure 459. Changes to the Report's Table

Format Data Conditionally

The last action we will perform on the table is to write an XPath expression to display the 'R' rating for movies in a bold red.

23. Click the **rating** glyph.
 24. In the **Properties** window, expand the **Default Color** property.
 25. Click the **XPath** field, and then click the "more" button (...).
- Stylus Studio displays the XPath Editor dialog box.
26. Type the following XPath expression:

```
if (rating = 'R') then 'red' else ""
```

Notice Stylus Studio's Sense:X auto-completion and text coloring as you type.

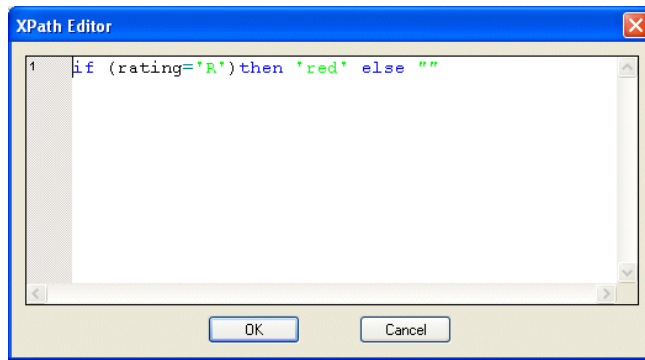


Figure 460. XPath Editor Dialog Box

27. Click OK.
28. Next, use the same process to enter this XPath expression for the **Bold** property:
if (rating = 'R') then 'bold' else ""
29. Click **Preview** (▶).

Our report now looks like this:

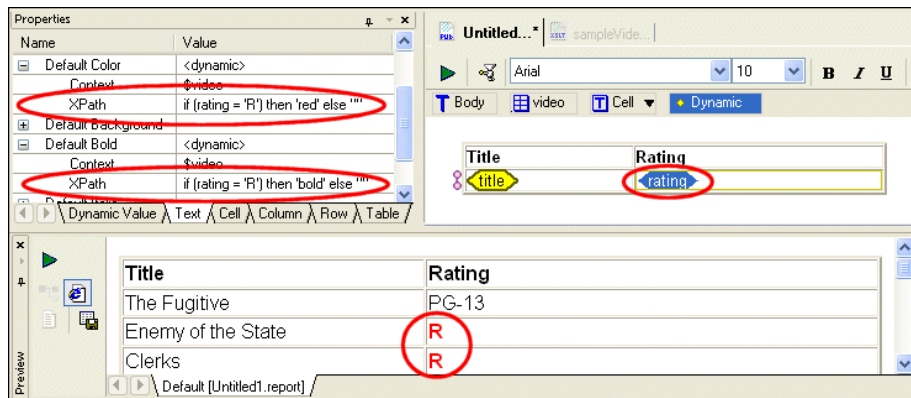


Figure 461. Finished Example Report

Generate the Code

Once the report is finished, we can generate XSLT or XQuery code to produce a report in HTML+CSS or XSL-FO formats.

30. Click the **Generate** button on the toolbar.

Stylus Studio displays the **Generate Transformation** dialog box.

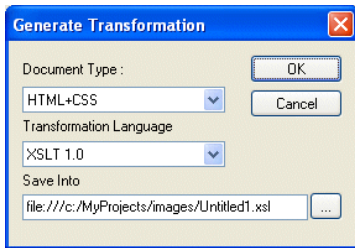


Figure 462. Generate Transformation Dialog Box

We want to use XSLT to generate our report in HTML+CSS, so we do not need to change the values for **Document Type** or **Transformation Language**.

31. The default name for the .xsl file is based on the .report file name. We change Untitled1.xsl to myMovieRatings.xsl and click **OK**.

Stylus Studio opens the generated XSLT in the XSLT Editor, as shown in [Figure 463](#).

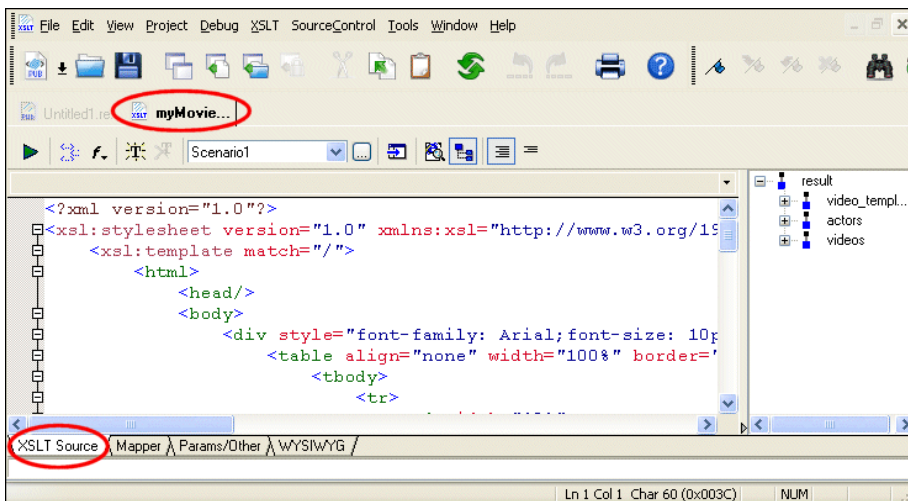


Figure 463. XSLT Generated from the XML Publisher Report

32. If we preview the XSLT, we see the same results as when we previewed the report in XML Publisher.

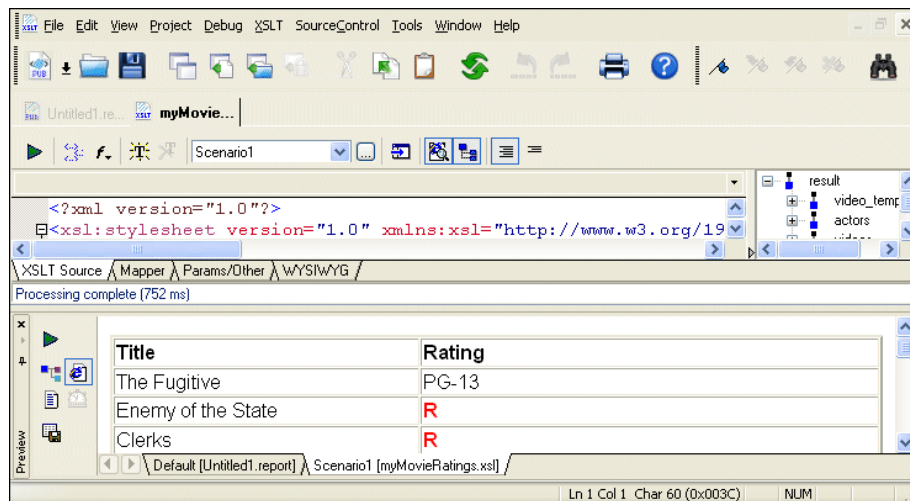


Figure 464. Preview of the XSLT Generated by XML Publisher

Properties Reference

This section contains reference information for the report component properties displayed in the **Properties** window. Where appropriate, properties are grouped by component – the section “[Table Properties](#)” on page 1001 has subsections for components you would typically associate with a table – sections for row and column properties, for example. The text property is associated with many components, as well as being a component in its own right. It has its own subsection (“[Text Properties](#)” on page 1003).

This section is organized as follows:

- “[Context and XPath Sub-Properties](#)” on page 1000
- “[Body Properties](#)” on page 1000
- “[Table Properties](#)” on page 1001
- “[List Properties](#)” on page 1003
- “[Text Properties](#)” on page 1003
- “[Repeater Properties](#)” on page 1004
- “[If Properties](#)” on page 1005
- “[Image Properties](#)” on page 1005

Context and XPath Sub-Properties

All of the properties in this section have Context and XPath sub-properties. They are described in “[Component Properties](#)” on page 980. In addition, some properties (ifs and repeaters, for example) have sub-properties that are unique to them. Unique sub-properties are described with the property to which they pertain.

Body Properties

Body properties affect all other components in an XML Publisher report unless they have their own format settings specified for them. Body properties are displayed on two tabs – **Document** and **Text**.

Table 150. Document Properties

<i>Property</i>	<i>Description</i>
Base URL	Base URL used for HTML links and image resolution.
Background	Sets the background color for the report.

Table 151. Text Properties

<i>Property</i>	<i>Description</i>
Alignment	Aligns body content along left margin, right margin, or in the center. Default is None.
Font	Sets the font for body contents. If no value is specified, Stylus Studio uses the font specified in the toolbar.
Size	Sets the font size for body contents. If no value is specified, Stylus Studio uses the font size specified in the toolbar.
Color	Sets the font color for body contents. If no value is specified, Stylus Studio uses the value specified by the Foreground Color tool in the toolbar.
Background	Sets the background color for body contents. If no value is specified, Stylus Studio uses the value specified on the Document tab.
Bold	Sets the body content to bold. If no value is specified, Stylus Studio uses the value specified by the Bold tool in the toolbar.

Table 151. Text Properties

<i>Property</i>	<i>Description</i>
Italic	Sets the body content to italic. If no value is specified, Stylus Studio uses the value specified by the Italic tool in the toolbar.
Underline	Sets the body content to underline. If no value is specified, Stylus Studio uses the value specified by the Underline tool in the toolbar.
Base URL	Base URL used for HTML links and image resolution.

Table Properties

See also:

- [“Row Properties”](#) on page 1002
- [“Column Properties”](#) on page 1002
- [“Cell Properties”](#) on page 1002
- [“Text Properties”](#) on page 1003

Table 152. Table Properties

<i>Property</i>	<i>Description</i>
Component Name	The name of the repeating element on which the table is based. This name appears in the navigation bar glyph that represents the table. Editable.
Width	The table width. If expressed as a percent, the table is drawn relative to the available page. If expressed as a number, the value is interpreted in points (pt).
Border	The width of the line used to draw the table border, in points (pt).
CellSpacing	The amount of space between cells, in points (pt).
CellPadding	The amount of space between a cell border and its contents, in points (pt).
Alignment	Aligns cell contents along left border, right border, or in the center. Default is None.

Row Properties

Table 153. Row Properties

<i>Property</i>	<i>Description</i>
Loop	A group of properties (Context, XPath, and Sort) that together describe how to iterate over a node set. See “ Component Properties ” on page 980.
Sort	A sub-property of Loop that allows you to specify, using an XPath expression, for example, how to sort table rows. If no value is specified, rows are displayed in document order.
Height	The row height in points (pt).
Background	The color of the row background. (Set the color of row values using the Text tab.

Column Properties

Table 154. Column Properties

<i>Property</i>	<i>Description</i>
Width	The column width in points (pt).
Background	The color of the column background.

Cell Properties

Table 155. Cell Properties

<i>Property</i>	<i>Description</i>
Background	The color of the cell background.

List Properties

See also “[Item Properties](#)” on page 1003.

Table 156. List Properties

<i>Property</i>	<i>Description</i>
Component Name	The name of the repeating element on which the list is based. This name appears in the navigation bar glyph that represents the list. Editable.
List Type	The type of symbol or character you want to use for the list: disk (the default), circle, decimal, lower-alpha, none, square, and upper-alpha.

Item Properties

Item properties are the same as Text properties. See “[Text Properties](#)” on page 1003.

Text Properties

Text properties are applicable to text components (**Insert > Text**) as well as to text in other components (tables, lists, repeaters, and ifs).

Table 157. Body Properties

<i>Property</i>	<i>Description</i>
Alignment	Aligns text along left margin, right margin, or in the center. Default is None.
Font	Sets the font for the selected text. If no value is specified, Stylus Studio uses the font specified in the toolbar.
Size	Sets the font size for the selected text. If no value is specified, Stylus Studio uses the font size specified in the toolbar.
Color	Sets the font color for the selected text. If no value is specified, Stylus Studio uses the value specified by the Foreground Color tool in the toolbar.

Table 157. Body Properties

<i>Property</i>	<i>Description</i>
Background	Sets the background color for the selected text. If no value is specified, Stylus Studio uses the value specified by the Background Color tool in the toolbar.
Bold	Sets the selected text to bold. If no value is specified, Stylus Studio uses the value specified by the Bold tool in the toolbar.
Italic	Sets the selected text to italic. If no value is specified, Stylus Studio uses the value specified by the Italic tool in the toolbar.
Underline	Sets the selected text to underline. If no value is specified, Stylus Studio uses the value specified by the Underline tool in the toolbar.

Repeater Properties

See also [“Text Properties”](#) on page 1003.

Table 158. Repeater Properties

<i>Property</i>	<i>Description</i>
Loop	A group of properties (Context, XPath, and Sort) that together describe how to iterate over a node set. See “Component Properties” on page 980.
Sort	A sub-property of Loop that allows you to specify, using an XPath expression, for example, how to sort the items in the repeater. If no value is specified, items are displayed in document order.
Direction	The way in which the data values that make up the repeater’s items will be added to the report – vertical (in a list), or horizontal (in a row). The default is Vertical .

If Properties

See also “Text Properties” on page 1003.

Table 159. If Properties

<i>Property</i>	<i>Description</i>
Condition	An XPath expression used to define the condition. Actions based on this condition are specified on the If glyph’s true and false tabs.

Image Properties

Table 160. Image Properties

<i>Property</i>	<i>Description</i>
Width	The image width in points (pt).
Height	The image height in points (pt).
Source	The location of the image file to be displayed in the report. This can be an absolute path, or a relative path specified in conjunction with the Body component’s Base URL property.
Alignment	Aligns the image relative to the page – left, right, or in the center. Default is None.

Chapter 14 **Integrating with Third-Party File Systems**



Integration with TigerLogic XDMS is available only in Stylus Studio XML Enterprise Suite.

Stylus Studio is fully integrated with Raining Data[®] TigerLogic[®] XML Data Management Server (TigerLogic XDMS).

This chapter describes how to work with this file system in Stylus Studio.

Using Stylus Studio with TigerLogic XDMS



Integration with the TigerLogic XDMS file system is available only in Stylus Studio XML Enterprise Suite.

This section describes how to use Stylus Studio with the TigerLogic XDMS file system and covers the following topics:

- [“Overview”](#) on page 1008
- [“Connecting to TigerLogic XDMS”](#) on page 1009
- [“Using Documents Stored on TigerLogic XDMS”](#) on page 1011
- [“Creating Collections”](#) on page 1012

Overview

Stylus Studio can read and write well-formed XML documents (.xml, .xsl, and so on) from/to TigerLogic XDMS collections. In Stylus Studio, you reference external files using a URL. The URL used to access TigerLogic XDMS files is `tig://server:port/database_name/collection_name/file_name.xml`:

- `tig` is the prefix that specifies that files will be read from TigerLogic XDMS
- `server:port` is the server name that hosts the TigerLogic XDMS file system; the port is the port used to connect to the server
- `database_name` is the name of the TigerLogic XDMS database you want to access
- `collection_name` is the name of the TigerLogic XDMS collection associated with the document you want to open
- `file_name.xml` is the name of the XML document you wish to access within that collection

For example, to access the `auction.xml` document in `mycollection`, you might enter the following in the **URL** field of the Stylus Studio **Open** dialog box:

```
tig://ntstylus-dev:3408/mydb/mycollection/auction.xml
```

Note Stylus Studio treats collections very much like directories, and the XML documents stored within a collection are treated like files within a directory.

You can save files back to the same collection from which they were read, or to some other file system (your local machine, for example). You can create new collections.

Note You can use a TigerLogic XDMS URL to open an XML document anywhere you can specify URLs in Stylus Studio. For example, you can use the TigerLogic XDMS URL to specify the source document for XQuery Mapper and then process the XQuery using the TigerLogic XDMS XQuery processor.

TigerLogic XDMS Version Support

Stylus Studio supports TigerLogic XDMS Version 2.6.

Connecting to TigerLogic XDMS

The TigerLogic XDMS file system is accessible from the

- **Open** dialog box
- **File Explorer** window

In addition, if you know the file system URL for a specific TigerLogic XDMS database, you can enter it in a **URL** field (in the **Open** dialog box, for example) and connect to the file system in that fashion.

What Happens When You Connect

When you connect to the server hosting TigerLogic XDMS, Stylus Studio caches the information used to establish the connection – host name and port, username, and password – for the duration of the Stylus Studio session. When you exit Stylus Studio, all connections are dropped. Only the host name and port information is retained to help simplify subsequent connections.

Note If a document in Stylus Studio (an XQuery or XSLT, for example) uses a document stored in a TigerLogic XDMS collection, when you open that document in a subsequent session, Stylus Studio automatically prompts you for the TigerLogic XDMS connection information.

How to Connect to TigerLogic XDMS

◆ **To connect to TigerLogic XDMS:**

1. In the **File Explorer** window, expand the TigerLogic XDMS folder.

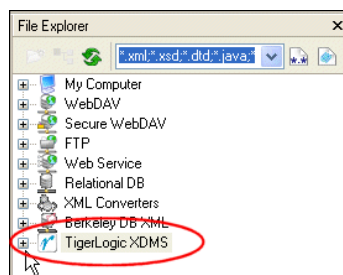


Figure 465. TigerLogic XDMS Icon in File Explorer

Alternative: On the Stylus Studio menu, Click **File > Open**, and then click the TigerLogic XDMS icon.



Figure 466. TigerLogic XDMS Icon in Open Dialog Box

Stylus Studio displays the **Choose Server** dialog box.

2. Complete the **Server Name**, (server address and port; myServer:3008, for example) **User Name**, and **Password** fields, and click OK.

Stylus Studio connects you to the host specified in the **Server name** field. Icons representing the host and the TigerLogic XDMS databases residing on that host appear in the **File Explorer** window.

3. Optionally, expand the database folders to view the TigerLogic XDMS collections stored on that database.
4. Optionally, expand the collection folders to view the XML documents stored there.

Reconnecting

If you lose your connection with the server hosting the TigerLogic XDMS file system, and a document from a TigerLogic XDMS collection is open, Stylus Studio displays the **Authentication Required** dialog box the next time you try to access that document.

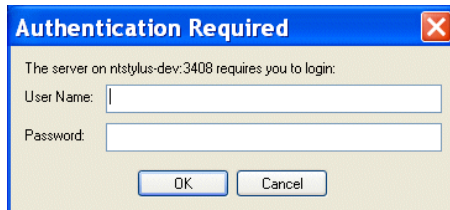


Figure 467. Authentication Required Dialog Box

This dialog box can appear, for example, when you

- Re-start Stylus Studio
- Refresh an XSLT or XQuery that uses an XML document from a TigerLogic XDMS collection

The **Authentication Required** dialog box prompts you for the username and password you used when you first established a connection with the server.

Using Documents Stored on TigerLogic XDMS


XML documents stored on the TigerLogic XDMS file system are available to you in Stylus Studio once you establish a connection to the server hosting TigerLogic XDMS, as described in [“Connecting to TigerLogic XDMS”](#) on page 1009.

Opening Documents

You can open a document stored in the TigerLogic XDMS file system by dragging it from the **File Explorer** window into an open document editor, the document editor tab area, or some other target. See [“Dragging and Dropping Files in the Stylus Studio”](#) on page 98 and [“Using the File Explorer”](#) on page 95 for more information. You can also select a document using the **Open** dialog box (**File > Open**).

Saving Documents

When you save a document to the TigerLogic XDMS file system, TigerLogic XDMS first validates the document to ensure that it is well-formed XML. If the document is well-formed, the document is saved; otherwise, you receive an error from the TigerLogic XDMS server.

Tip Use the Stylus Studio well-formedness checker  to check your XML documents before saving them to TigerLogic XDMS.

Auto-Save and Backup Files

Stylus Studio has an option (select **Tools > Options** from the Stylus Studio menu) that automatically saves modified documents at an interval you determine (every 10 minutes is the default). This option is off by default.

The creation of backup files is managed by the TigerLogic XDMS file system. Stylus Studio does not create .bak files of files saved to the TigerLogic XDMS file system.

Using Documents in XQuery and XSLT

You can use XML documents stored in the TigerLogic XDMS file system as source and target documents in Stylus Studio XQuery and XSLT. They are treated as any other document with one exception: when you open a document in Stylus Studio that uses a document stored in the TigerLogic XDMS file system, Stylus Studio prompts you to supply authentication information (username, password) before reestablishing connection with the server hosting the TigerLogic XDMS file system.

Creating Collections

You can create new TigerLogic XDMS collections in Stylus Studio.

◆ **To create a TigerLogic XDMS collection:**

1. Connect to a TigerLogic XDMS server, as described in “[Connecting to TigerLogic XDMS](#)” on page 1009.
2. Open the **File Explorer** window if it is not already open (**View > File Explorer**).
3. Right-click the TigerLogic XDMS server instance in which you wish to create the collection.
4. Select **New Folder** from the short-cut menu.
A new folder icon appears under the icon for the TigerLogic XDMS server instance with the default name, *New Folder*.
5. Edit the default collection name and press Enter.

Alternative:

You can also create a new TigerLogic XDMS collection from the Open dialog box.

1. Connect to a TigerLogic XDMS server, as described in “[Connecting to TigerLogic XDMS](#)” on page 1009.
2. Click the **New Folder** button on the **Open** dialog box.

Chapter 15 **Extending Stylus Studio**

Stylus Studio provides several ways to extend its native functionality. This chapter describes features that allow you to specify other XML validation engines, and features that allow you to define and register custom document wizards.

This chapter covers the following topics:

- [“Custom XML Validation Engines”](#) on page 1014
- [“Custom Document Wizards”](#) on page 1020
- [“Custom File Systems”](#) on page 1031

Custom XML Validation Engines

Stylus Studio supports several XML validation engines, including the MSXML4.0 SAX Parser, Xerces-J 2.3.0, and .NET XML. The custom validation engine feature lets you register your own XML validation engine with Stylus Studio. Custom validation engines are added to the **Validate Document** drop-down list in the XML Editor once you register them with Stylus Studio, as shown in [Figure 468](#).

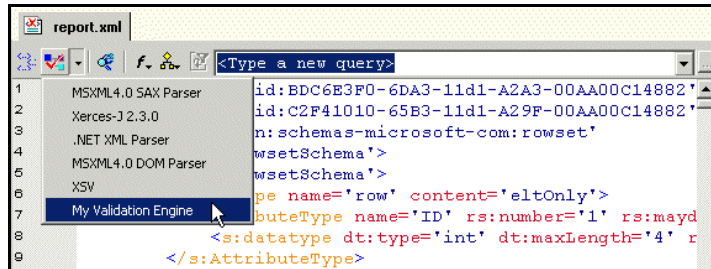


Figure 468. Validate Document Drop-Down List

Output for custom validation engines is displayed in Stylus Studio’s **Output Window**. Output for other custom applications, such as that created by the custom document wizard, is also displayed in Stylus Studio’s **Output Window**.

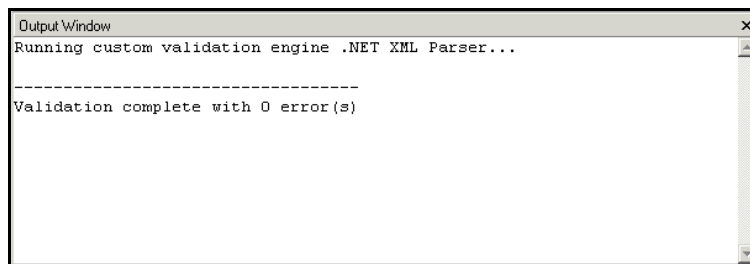


Figure 469. Output Window

Registering a Custom Validation Engine

The process of registering a custom validation involves the following steps:

1. Make the necessary custom validation engine available to Stylus Studio.

2. Configure the custom validation engine on the **Custom Validation Engines** page of the **Options** dialog box. This step involves
 - a. Providing a name.
 - b. Specifying a command line template.
 - c. Defining any arguments required by the command line.
 - d. Optionally specifying the initial directory, path, and classpath to be used by the custom validation engine.
 - e. Optionally setting a feature that prompts the custom validation engine user for arguments when the custom validation engine is run.

More information for each of these steps is provided in the following section, [“Configuring a Custom Document Wizard”](#) on page 1021.

Configuring a Custom Validation Engine

This section provides information and procedures for configuring a custom validation engine. It covers the following topics:

- [The Custom Validation Engines Page](#) on page 1016
- [How to Configure a Custom Validation Engine](#) on page 1018

The Custom Validation Engines Page

You use the **Custom Validation Engines** page of the **Options** dialog box to work with custom validation engines in Stylus Studio.

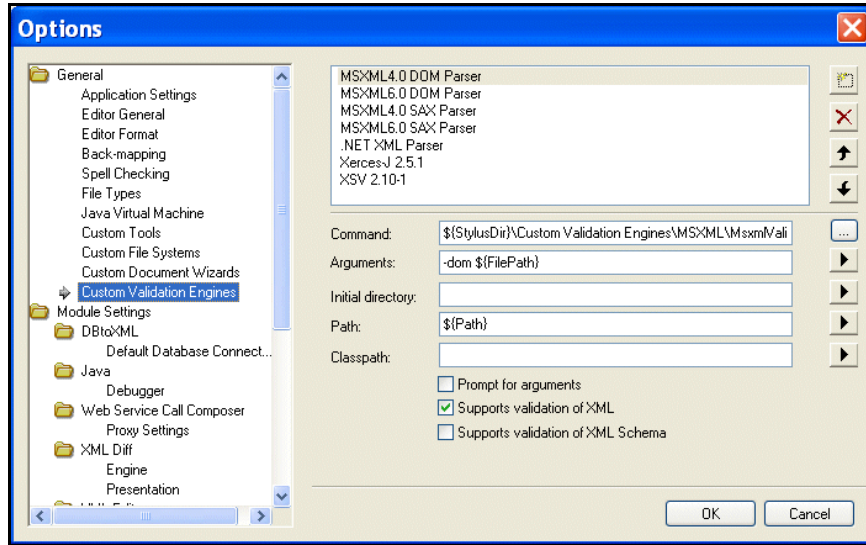


Figure 470. Custom Validation Engines Page


How to display

◆ **To display the Custom Validation Engines page:**

1. In the Stylus Studio menu bar, select **Tools > Options**.
The **Options** dialog box appears.
2. If necessary, expand **Application Settings** and click **Custom Validation Engines**.
The **Custom Validation Engines** page appears.

About macros


Stylus Studio provides macros for some fields to help speed creation of custom validation engines. Any macro you use to configure the custom validation engine is resolved when it is run.

Available macros vary based on the field for which they are being used. To display macros available for a given field, click . Predefined macros include

- `{{FilePath}}` – The complete path of the XML file to be validated.

- `${FileDir}` – The directory in which the XML file to be validated is stored.
- `${FileName}` – The name of the XML file to be validated.
- `${FileExt}` – The extension of the XML file to be validated.
- `${ClassPath}` – The Classpath environment variable.
- `${StylusDir}` – The path of the Stylus Studio installation directory.

Name

When you click the **New** button () to create a new custom validation engine, Stylus Studio displays an entry field for the name.

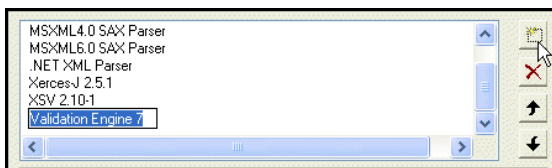


Figure 471. Specifying a Custom Validation Engine Name

You should replace the default name (**Validation Engine 1**, for example) with the name you want to associate with the custom validation engine. The name you enter is displayed in the drop-down in the XML Editor.

Custom validation engines are displayed in the **Validate Document** drop-down list in the order in which they appear here.


◆ You can change the custom validation engine order by

1. Selecting the custom validation engine whose order in the list you want to change.
2. Clicking the up or down arrow to the right of the custom validation engine list box as needed.


Command

You use the **Command** field to specify the command line used to invoke the custom validation engine. This is typically the path to the `.exe`, `.cmd`, or `.bat` file that starts the application.


Arguments

You use the **Arguments** field to specify any arguments required by the custom validation engine. Click  to browse predefined macros.


Initial Directory

You use the **Initial directory** field to specify the directory you want Stylus Studio to use as the current directory when the custom validation engine is run. Click  to browse predefined macros.

Path

You use the **Path** field to define paths to any files (such as .exe and .dll) required by the custom validation engine. You do not have to define any paths that are already defined in your PATH environment variable. Separate multiple paths with a semicolon. Click  to browse predefined macros.

Classpath

You use the **Classpath** field to define paths to any JVM files required by the custom validation engine (such as .jar and .class). You do not have to define any paths that are already defined in your PATH environment variable. Click  to browse predefined macros.

Prompt for arguments

The **Prompt for arguments** feature displays a dialog box when the custom validation engine is run.

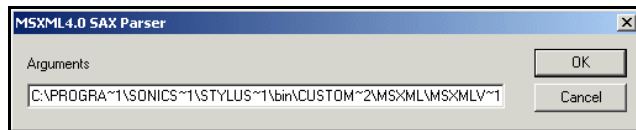


Figure 472. Argument Prompt

The **Arguments** field allows the user to change the command line and arguments configured with the custom validation engine when it was registered with Stylus Studio.

How to Configure a Custom Validation Engine

Before performing this procedure, you should be familiar with the information in [“The Custom Validation Engines Page”](#) on page 1016.

◆ To configure a custom validation engine:

1. Display the **Custom Validation Engines** page of the **Options** dialog box. See [“How to display”](#) on page 1016 if you need help with this step.

2. Click the **New** button and enter a name for the custom validation engine. Remember that this value is displayed in the **Validate Document** drop-down list in the XML Editor.
3. Specify the command line any required arguments. See [“Command”](#) on page 1017 and [“Arguments”](#) on page 1017 if you need help with this step.
4. Optionally, specify an initial directory, path and classpath.
5. Click **Prompt for arguments** if you want Stylus Studio to display a dialog box that allows the user to change the command line or arguments when the custom validation engine is run.
6. Click **OK**.

Custom Document Wizards

Stylus Studio’s *custom document wizard* feature allows you to create and configure document wizards that invoke third-party file conversion and document generation tools, such as Thai Open Source’s Trang. When run, a custom document wizard passes argument values provided by the user (the name of the file to be converted, for example) to the command line that invokes the third-party tool. The third-party tool generates an output file as specified in the custom document wizard’s command line, and the file is then opened by Stylus Studio in the appropriate editor.

An example of a custom document wizard is the DTD to XML Schema (Trang) document wizard shipped with Stylus Studio.

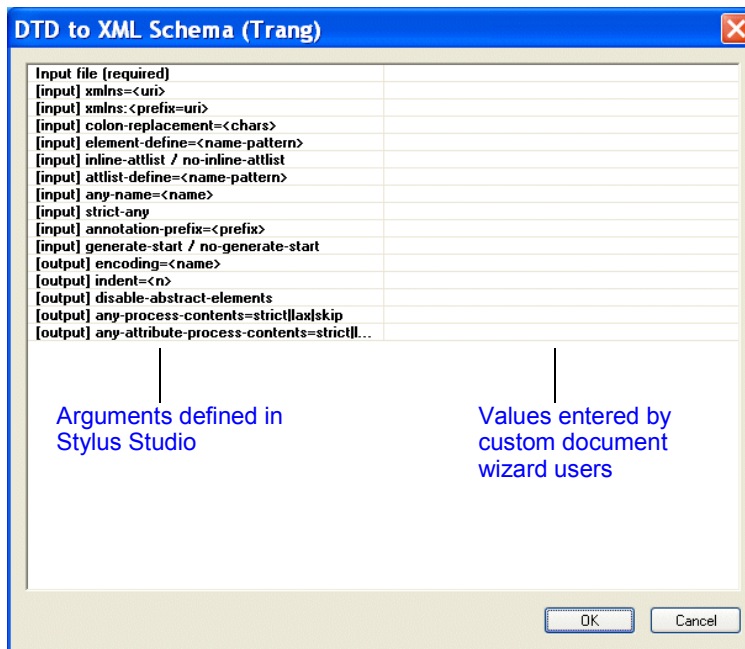


Figure 473. A Custom Document Wizard

This document wizard was created using the custom document wizard feature.

Registering a Custom Document Wizard

The process of registering a custom document wizard in Stylus Studio involves the following steps:

1. Make the necessary third-party software available to Stylus Studio. For example, any .jar or .exe files associated with the document conversion or generation tool must be accessible from the Stylus Studio installation.
2. Configure the custom document wizard on the **Custom Document Wizards** page of the **Options** dialog box. This step involves
 - a. Providing a name and, optionally, an icon, for the custom document wizard.
 - b. Setting the document type.
 - c. Specifying a command line template.
 - d. Defining any arguments required by the command line.
 - e. Optionally setting a trace feature that displays processing provided by the third-party tool.

More information for each of these steps is provided in the following section, [Configuring a Custom Document Wizard](#) on page 1021.

Configuring a Custom Document Wizard

This section provides information and procedures for configuring a custom document wizard. It covers the following topics:

- [“The Custom Document Wizards Page”](#) on page 1022
- [“Defining Arguments”](#) on page 1026
- [“How to Configure a Custom Document Wizard”](#) on page 1030

The Custom Document Wizards Page

You use the **Custom Document Wizards** page of the **Options** dialog box to work with custom document wizards in Stylus Studio.

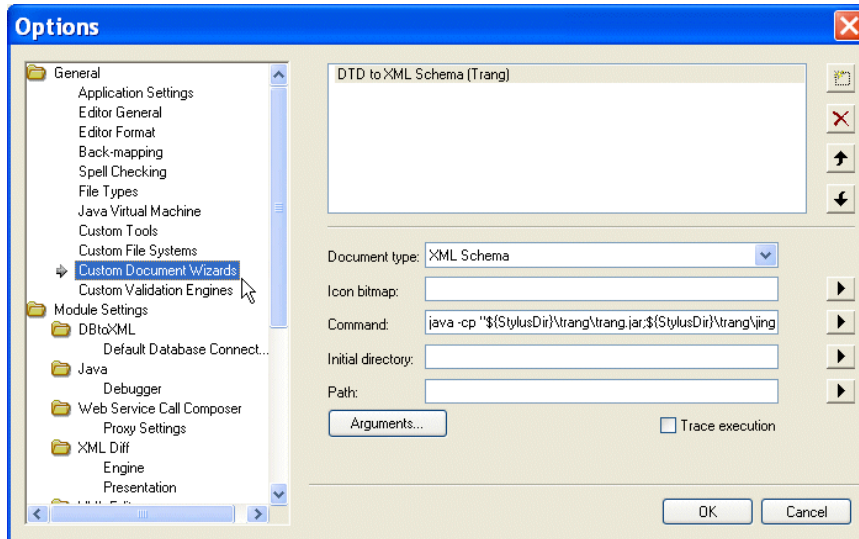



Figure 474. Custom Document Wizards Page

This section describes how to display the **Custom Document Wizards** page and information about its fields.

How to display

- ◆ **To display the Custom Document Wizards page:**
 1. In the Stylus Studio menu bar, select **Tools > Options**.
The **Options** dialog box appears.
 2. If necessary, expand **Application Settings** and click **Custom Document Wizards**.
The **Custom Document Wizards** page appears.

About macros


Stylus Studio provides macros for some fields to help speed creation of custom document wizards. Available macros vary based on the field for which they are being used. To display macros available for a given field, click .

Predefined macros include

- `#{StylusDir}`, which indicates that the path you are specifying is relative to the Stylus Studio installation directory.
- `#{PATH}`, which specifies the PATH environment variable.
- `#{OutputFile}`, which is used to specify the output generated by the document wizard. This macro is available in the **Command** field only.

In addition, Stylus Studio creates argument variable macros for any arguments you define and displays them with other **Command** field macros.

Name

When you click the **New** button () to create a new custom document wizard, Stylus Studio displays an entry field for the name.

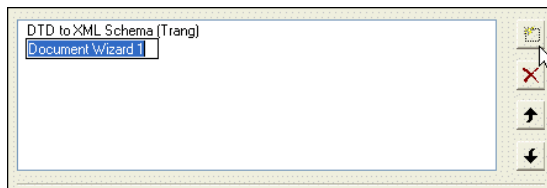


Figure 475. Specifying a Custom Document Wizard Name

You should replace the default name (DocumentWizard1, for example) with the name you want to associate with the custom document wizard. The name you enter is

- Displayed in the **Document Wizards** dialog box along with the icon you specify for the custom document wizard.
- Used in the title bar of the dialog box the user sees when running the custom document wizard.

Document type

The **Document type** field displays a drop-down list of available document types when you click it:

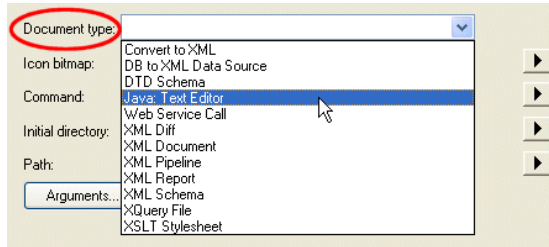



Figure 476. Document Type Field

The document type is the type of output generated by the custom document wizard (XML Schema, XQuery, and so on). The value you select determines

- The tab in the **Document Wizards** dialog box on which the custom document wizard is displayed (**XML Editor**, **XSLT Editor**, or **Java**, for example)
- The editor Stylus Studio uses to display the output generated by the document wizard

Icon bitmap

You use the **Icon bitmap** field to specify the path for the icon you want to represent the custom document wizard. This icon, along with the name you give the custom document wizard, is displayed in the **Document Wizards** dialog box. Click  to browse for the file you want to specify or to insert the `${StylusDir}` macro.

If you leave the **Icon bitmap** field blank, Stylus Studio uses the following default icon for the custom document wizard: 

Command line

You use the **Command line** field to specify a command line template. Stylus Studio uses this template to compose the command line that invokes the custom document wizard. Variables, such as `${InputFile}`, are used in place of actual arguments. Users specify argument values when they run the custom document wizard.

Consider the following example:

```
java -cp my.jar com.exln.stylus.Import ${QuoteChar} ${InputFile} ${OutputFile}
```

This command line template allows Stylus Studio to start the specified Java class with a command line that includes the `QuoteChar`, `InputFile`, and `OutputFile` arguments.

Argument variables can appear anywhere in the command. They must be in the form `${name}`. For example:


```
`${InputFile}  
`${OutputFile}  
`${LoggingOption}  
`${SomeArgument}
```

You must specify the `${OutputFile}` argument variable in every command line template. Stylus Studio always generates the name of the file it opens as the value for the `${OutputFile}` argument variable.

As with the **Icon bitmap** field, you can click  to display a menu that provides shortcuts that help you specify the command line template. This menu lets you


- Display the **Custom Document Wizard Arguments** dialog box, in which you can specify the command line arguments and their properties. See [“Defining Arguments”](#) on page 1026 for more information.
- Browse for and select a file you want to specify.
- Insert the `${StylusDir}` macro.
- Insert argument variable macros for arguments you have already defined.

Initial directory

You use the **Initial directory** field to specify the directory you want Stylus Studio to use as the current directory when the custom document wizard is run. Click  to browse for the file you want to specify or to insert the `${StylusDir}` macro.

Path

You use the **Path** field to define paths to any files required by the custom document wizard. You do not have to define any paths that are already defined in your `PATH` environment variable. Separate multiple paths with a semicolon.

Click  to display a menu that provides shortcuts that help you specify the `PATH` field. From this menu you can

- Browse for and select a file you want to specify.
- Insert the `${StylusDir}` macro.
- Insert the `${PATH}` macro.

Trace execution

If the custom document wizard you are configuring outputs processing information (error messages, stack traces, and so on), you can use the **Trace execution** feature to display this information in the **Output Window** of the Stylus Studio editor used to display the custom document wizard's generated document.

Defining Arguments

You must define any arguments required by the custom document wizard using the **Custom Document Wizard Arguments** dialog box.

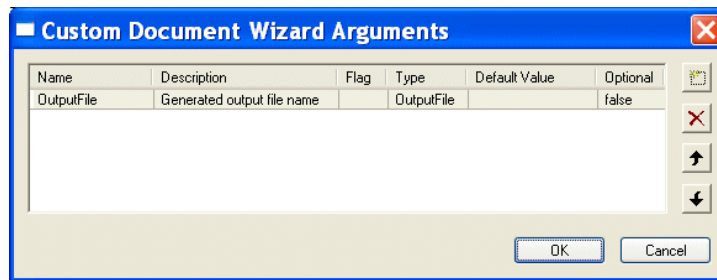


Figure 477. Custom Document Wizard Arguments Dialog Box

Stylus Studio uses the arguments you define here to


- Compose the dialog box used to run the custom document wizard. That dialog box enables users to provide values for the arguments you define.
- Create argument variable macros, which you can then use to compose the command line template. Stylus Studio displays the argument variable macro it creates in the **Command** field menu.

Note Every variable used in the command line template must be defined in the **Custom Document Wizard Arguments** dialog box.

How to display

There are two ways to display the **Custom Document Wizard Arguments** dialog box:

- Click the **Arguments** button on the **Custom Document Wizard page**. Use this procedure if you want to define arguments *before* composing the command line template.

- Select **Edit Arguments** from the **Command** field menu that is displayed when you click . You can use this procedure if you want to define arguments while composing the command line.

OutputFile argument

Stylus Studio creates an OutputFile argument for each custom document wizard. You cannot delete this argument. You can change its order, if necessary, as described in the following section.

Argument order

By default, the arguments you define in the **Custom Document Wizard Arguments** dialog box are displayed to users in the order in which they are created. Arguments are displayed in a simple two-column grid, with the argument description in the first column, and an entry field for the argument value in the other. (See [Figure 473](#) for an illustration of a custom document wizard dialog box.)

Also by default, the OutputFile argument appears first.

◆ You can change the argument order by

1. Selecting the argument whose order you want to change.
2. Clicking the up or down arrow to the right of the argument list box as needed.

Note Whether or not the argument order defined here has to match the argument order in the command line template will vary from one custom document wizard to the next – arguments for some applications can be order independent, for example. Generally speaking, it is good practice for the argument order in the **Custom Document Wizard Arguments** dialog box to match that in the command line template.

Argument attributes

You can specify the following attributes for each argument you define:

- **Name.** Stylus Studio uses the value you enter in the **Name** field to compose the argument variable macro. This name is not displayed to custom document wizard users. Required.
- **Description.** The value you enter in the **Description** field appears in the custom document wizard dialog box that is displayed to users when they run the wizard. The description should provide users with adequate information about the argument's

expected value. It can be useful to distinguish input and output arguments, for example. Required.

- **Flag.** The flag associated with the argument (-v, or simply - or /, for example). When Stylus Studio composes the command line for the custom document wizard, it uses the flag value as a prefix to the argument value supplied by the user.

Note

The **Flag** field must be specified for Boolean arguments.

- **Type.** The argument's data type. [Table 161](#) summarizes valid values for the **Type** field and describes possible values for those types

Table 161. Type Field Values

<i>Type</i>	<i>Description</i>
boolean	The value for a Boolean argument must be true or false. If the value is true, Stylus Studio inserts the value of the associated Flag attribute in the command line. No value other than the Flag value appears in the command line for Boolean arguments. If the value is false, the associated Flag value does not appear in the command line. If you set Type to boolean, you must specify the argument's Flag attribute.
InputFile	The value for an InputFile argument is a URL that the custom document wizard user enters or selects by clicking the Browse button. If the format of the URL is for a protocol other than the file protocol, Stylus Studio copies the file into a temporary local file and uses the name of the temporary local file in the command line. You can specify multiple arguments whose data type is InputFile.
OutputFile	The custom document wizard user does not specify a value for the OutputFile argument. Exactly one argument must be of the OutputFile type. Stylus Studio generates a value for the OutputFile argument and inserts it in the command line.
string	The value for a string argument can be anything specified by the custom document wizard user. Stylus Studio encloses the string values in quotation marks when composing the command line.


- **Default Value.** The value used by Stylus Studio for optional arguments, unless another value is specified by the user when the custom document wizard is run. Default values

for required arguments are ignored – Stylus Studio requires users to enter values for required arguments.

- Optional. Whether or not the argument is optional. Valid values for this field are true or false.

How to define an argument

◆ To define an argument:

1. Display the **Custom Document Wizard Arguments** dialog box. See “How to display” on page 1026 if you need help with this step.
2. Click the **New** button ().

A new argument is displayed in the **Custom Document Wizard Arguments** dialog box, with a default name and other default values.



Figure 478. Custom Document Wizard Arguments Dialog Box

3. Complete the argument attributes as described in earlier in this section. Remember that **Description** values appear in the custom document wizard dialog box when the user runs the wizard.
4. To define another argument, click the **New** button again.
5. If necessary, use the **Up** and **Down** arrows to change the argument order. Remember that the order in which arguments are displayed here is the order in which they appear in the custom document wizard dialog box when the user runs the wizard.
6. Click **OK**.

How to Configure a Custom Document Wizard

Before performing this procedure, you should be familiar with the information in [“The Custom Document Wizards Page”](#) on page 1022 and [“Defining Arguments”](#) on page 1026.

◆ **To configure a custom document wizard:**

1. Display the **Custom Document Wizards** page of the **Options** dialog box. See [“How to display”](#) on page 1022 if you need help with this step.
2. Click the **New** button and enter a name for the custom document wizard. Remember that this value is used as the title for the dialog box displayed to the user when they run the wizard, as well as for the label associated with the custom document wizard icon displayed in the **Document Wizards** dialog box.
3. Click the **Arguments** button and define the wizard’s arguments on the **Custom Document Wizard Arguments** dialog box. See [“Defining Arguments”](#) on page 1026 if you need help with this step.
4. Select the custom document wizard’s document type.
5. Specify the command line template. See [“Command line”](#) on page 1024 if you need help with this step.
6. Optionally, specify an initial directory and path.
7. Click **Trace execution** if you want to display processing information generated by the custom document wizard in the **Output Window** of the Stylus Studio editor window associated with the custom document wizard’s **Document type**.
8. Click **OK**.

Custom File Systems

Warning The Custom File System interface has been deprecated in Stylus Studio 2007 XML Enterprise Suite Release 2 and support for this feature will be removed in a future release. The ability to register a custom file system with Stylus Studio is currently preserved for backward compatibility only.

A *file system* is view that represents information as a series of files in at least one folder or directory. An individual file within that file system – such as an XML document, an XQuery program, or a DTD – can contain any text or binary information.

A *custom file system* is a way to extend the file/folder metaphor beyond traditional disk-based files (accessed with the `file:` scheme), and even beyond web-based files (WebDAV's `http:` and `https:`, and the `ftp:` scheme of the file transfer protocol) for example. Stylus Studio supports several standard and custom file systems, including Windows, WebDAV, and FTP.

These file systems are referred to as “custom” because, although some are packaged with Stylus Studio, you can create your own. For example, you could create an LDAP custom file system to query an LDAP directory as if it were a file system.

There is a Java API that allows you to access the custom file interface in Stylus Studio.

Tip The Stylus Studio interface to Tiger Logic XDMS was implemented using the File System Java API to create a custom file system.

Creating a Custom File System

The process of creating a custom file system in Stylus Studio involves two main steps:

1. Use the Stylus Studio File System Java API to create the Java wrapper class for your data source. See “[File System Interfaces](#)” on page 1032 for more information on this step.
2. Register the finished custom file system with Stylus Studio, to make it available through the Stylus Studio **Open**, **Save**, and **Save As** dialog boxes, as shown in

Figure 479. See “Registering a Custom File System” on page 1033 for more information on this step.

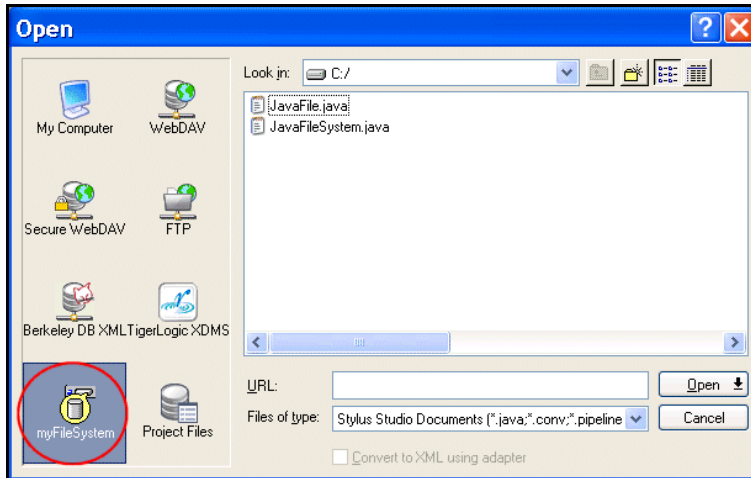


Figure 479. Custom File System Registered with Stylus Studio

File System Interfaces

The Stylus Studio file system interfaces are packaged in `CustomFileSystem.jar`, in the Stylus Studio `\bin` directory. To define a custom file system, you implement these interfaces:

- `com.exln.stylus.io.StylusFile`
- `com.exln.stylus.io.StylusFileSystem`

Once you have implemented the interfaces, compile them into your own `.jar` file (`myCustomFileSystem.jar`, for example), and register the custom file system with Stylus Studio.

Examples

Implementation examples of the `StylusFileSystem` and `StylusFile` interfaces, `JavaFileSystem.java` and `JavaFile.java`, are included in the `\examples\CustomFileSystem` directory where you installed Stylus Studio.

Registering a Custom File System

This section assumes you have already implemented the custom file system interfaces, compiled, and created a .jar file (myFileSystem.jar, for example.). You cannot complete the registration process unless the custom file system has been created.

This section covers the following topics:

- “[The Custom File Systems Page](#)” on page 1033
- “[How to Display](#)” on page 1034
- “[Fields](#)” on page 1034
- “[How to Register a Custom File System](#)” on page 1035

The Custom File Systems Page

You use the **Custom File Systems** page of the **Options** dialog box to register a custom file system.

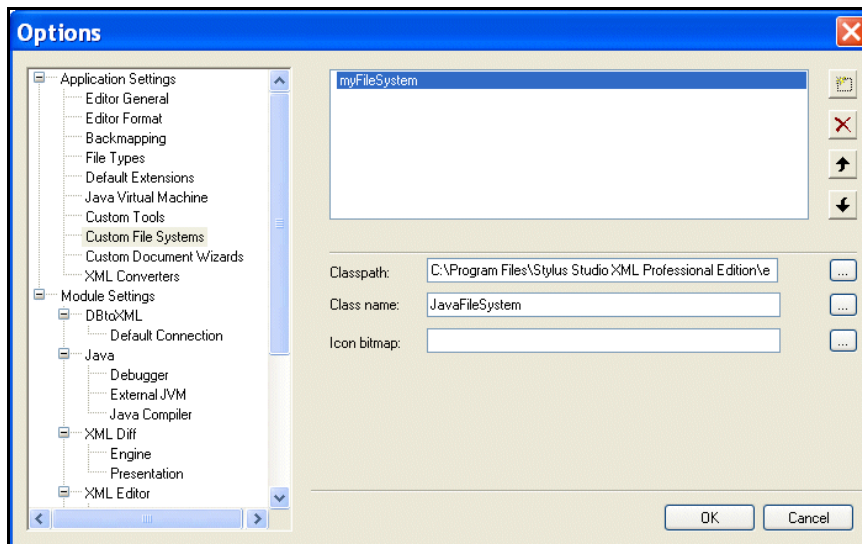


Figure 480. Options Dialog Box for Custom File Systems

How to Display


◆ **To display the Custom File Systems page:**

1. In the Stylus Studio menu bar, select **Tools > Options**.
The **Options** dialog box appears.
2. If necessary, expand **Application Settings** and click **Custom File Systems**.
The **Custom File Systems** page appears.

Fields

The **Custom File Systems** page of the **Options** dialog box has the following fields which you need to specify values for in order to register your custom file system.

Name

When you click the **New** button () to create a new custom file system, Stylus Studio displays an entry field for the name.

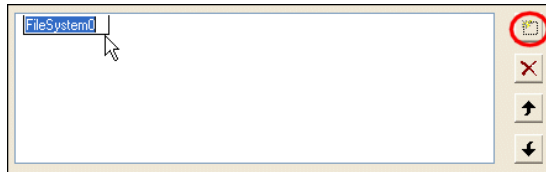


Figure 481. Naming a New Custom File System

You should replace the default name (**FileSystem0**, for example) with the name you want to associate with the custom file system. The name you enter is displayed in Stylus Studio **Open**, **Save**, and **Save As** dialog boxes.


Classpath

You use the **Classpath** field to specify the required classpath for the custom file system.

Class name

You use the **Class name** field to specify the name of the custom file system class created using the Stylus Studio custom file system Java API. If you use the **Browse** button, Stylus Studio filters the files displayed in the **Java Class Browser** dialog box to show only those classes that implement the `CustomFileSystem.jar` interfaces.

Icon bitmap

You use the **Icon bitmap** field to specify the icon to associate with the custom file system. This icon appears in the **Open**, **Save**, and **Save As** dialog boxes. If the **Icon bitmap** field is left empty, Stylus Studio uses a default icon ().

How to Register a Custom File System

◆ To register a custom file system:

1. Display the **Custom File Systems** page of the **Options** dialog box. See [“How to Display”](#) on page 1034 if you need help with this step.
2. Click the **New** button and enter a name for the custom file system. Remember that this value appears in the **Open**, **Save**, and **Save As** dialog boxes along with the icon you select.
3. Specify the classpath and class name.
4. Optionally, specify the icon you want to use to represent the custom file system.
5. Click **OK**.

Chapter 16 The Stylus Studio Java API

Note The material previously in this chapter has been deprecated in Stylus Studio 2007 XML Enterprise Suite Release 2. The functionality provided by the Stylus Studio Java API has been replaced by DataDirect XML Converters™ standalone components for Java™ and .NET. See the DataDirect XML Converters documentation for more information:
<http://www.xmlconverters.com/doc/>.

A

- ActiveX controls
 - xqDoc and 813
- alerts
 - for video demonstrations li
- ancestor axis 665
- ancestor-or-self axis 669
- AND operator 677
- annotating
 - XML pipelines and 911
- applying stylesheets
 - how it is done 333
 - Stylus Studio function key for 366
- Architecture data type property 300
- attribute axis 667
- attributes
 - matching template 356
- auto detect
 - feature for configuring Java components 136
- automatic tag completion
 - Sense:X 10
 - Stylus Studio feature for 10
- axis syntax in queries 663

B

- back-mapping
 - described 36
 - using in templates 499
 - XPath query results and 637

- XSLT processors that support 387
- backmapping
 - in XML pipelines 884
- backup copies of documents 212
- Base data type property 313
- BCD data type properties 300
- binary data type properties 301
- binary files
 - converting to XML 215
- BLOBs
 - querying 632
- bookmarks
 - setting 149
 - XQuery debugging and 791
- bookstore.xml 640
- boolean data type properties 301
- boolean() function 678
- Booleans
 - converting operands to 678
 - expressions 677
 - functions 678
- breakpoints
 - in stylesheets 494
 - in XQuery documents 788
- byte data type properties 304

C

- C Rules for Octal and Hex data type property 314

- call stack
 - displaying the Call Stack window in the XQuery editor 790
- canonical XML
 - converting XML to canonical XML 211
- case sensitivity
 - Boolean operators and 677
 - queries and 654
- `ceiling()` function 683
- chaining stylesheets 128
- character map
 - in XSLT 415
- checking spelling 152
- child axis 664
- classpath
 - setting for a project 108
- ClearCase
 - using with Stylus Studio 112, 114
- code
 - code folding 146
 - generating Java code for XQuery 819
 - generating Java code for XSLT 401
 - selecting lines in 149
- collection() functions
 - creating a collection() statement 782
 - creating database connections for 779
 - DataDirect XQuery processor and 778
 - handling invalid SQL characters in 779
 - querying relational data with 777
 - Stylus Studio built-in processor and 778
 - syntax 782
 - using 777
 - using in Stylus Studio 778
 - XQuery processors and 778
- collections
 - connecting to TigerLogic XDMS
 - collections 1008
 - creating in TigerLogic XDMS 1012
- collections. *See* collection() functions
- command line
 - custom document wizard arguments 1026
 - running Stylus Studio from 127
 - running XML diff 205
 - utilities 127
 - XML validation utility 130
 - XQuery utility 129
 - XSLT utility 128
- comma-separated files
 - see* CSV files
- `comment()` function 694
- Comp3 data type properties 305
- Comp3 properties for custom XML
 - conversions 305
- comparing node sets 685
- comparing XML documents 177
 - merged view 190
 - text view 189
 - tree view 188
- `concat` function blocks
 - in XQuery 775
- `concat()` function 673
- concatenating strings 673
- conditions
 - expressing
 - in XQuery Mapper 776
 - expressing in XSLT Mapper 475
- configuring
 - Java components 134
- `contains()` function 671
- context node 657
- context node set 657
- `.conv` files
 - default Stylus Studio module and 93
- Convert to XML
 - field names
 - display of 230
 - specifying field names in XML output 247
 - video demonstration 271
- Convert to XML Editor
 - changing fonts in 232
 - display features for document pane 231
 - displaying grid lines in 232
 - displaying pattern matches in 231
 - displaying the ruler in 231
 - Go To dialog box 233
- converter URL scheme
 - building a converter URL 288
 - displayed in Stylus Studio 287
 - parts of 284
 - syntax of 285
 - using with user-defined `.conv` files 285

- converters
 - for converting EDI 271
 - using in an XML pipeline 905
 - converting EDI to XML 271
 - creating custom message types 272
 - converting operands
 - to Booleans 678
 - to numbers 681
 - to strings 675
 - `count()` function 700
 - creating templates
 - example 358
 - how to 386
 - creating XQuery
 - using the XQuery Mapper 750
 - creating XSLT
 - using the XSLT Mapper 449
 - cross-language debugging in 916
 - cross-language debugging in XML pipelines 916
 - CSV files
 - converting to XML
 - Custom XML Conversions module 215
 - current context
 - number of nodes in 700
 - current node
 - definition 657
 - `current()` function 702
 - custom document wizards
 - about 1020
 - arguments 1026
 - command line arguments 1026
 - configuring 1021
 - defining arguments 1026
 - document types for 1024
 - how to configure 1030
 - macros for 1022
 - naming 1023
 - registering 1021
 - specifying a command line template for 1024
 - Custom Document Wizards page in Options
 - dialog box 1022
 - custom file systems
 - Custom File Systems page 1033
 - definition 1031
 - Java API for 1032
 - naming 1034
 - registering with Stylus Studio 1033
 - custom validation engines for XML
 - about 1014
 - configuring 1015
 - macros for 1016
 - naming 1017
 - registering 1014
 - Custom Validation Engines page in Options
 - dialog box 1016
 - custom XML conversion definitions
 - opening files with 268
 - custom XML conversions
 - creating a custom XML conversion definition 266
 - data type properties 299
 - URL scheme for 284
 - using with `converter` URLs 285
 - Custom XML Conversions module
 - video demonstration 215
 - custom XML Converters
 - URL scheme for 284
- ## D
- data sources
 - XML publisher reports and 945
 - data type properties 305
 - Architecture 300
 - Base 313
 - BCD properties for custom XML conversions 300
 - binary properties for custom XML conversions 301
 - boolean properties for custom XML conversions 301
 - byte properties for custom XML conversions 304
 - C Rules for Octal and Hex 314
 - common properties for custom XML conversion data types 299
 - Date Format 306
 - date properties for custom XML conversions 305
 - datetime properties for custom XML conversions 308

- Date-Time Separator 308
- Decimal 313
- decimal properties for custom XML
 - conversions 308
- double properties for custom XML
 - conversions 309
- Endian 309, 310, 311, 312, 316
- False Output As 302
- False Value Match List 302
- float properties for custom XML
 - conversions 310
- for custom XML conversions 299
- HMS Separator 319
- integer properties for custom XML
 - conversions 311
- Left Padding 303, 307, 315, 317, 319
- long properties for custom XML
 - conversions 312
- Lookup List 299
- Normalize White Space 317
- Notes 299
- number properties for custom XML
 - conversions 313
- Omit from Output 299
- Packed 300
- Rendering 301
- Right Padding 303, 307, 315, 317, 319
- Scaling Factor 300, 304, 305, 309, 310, 311, 312, 314, 316, 320
- short properties for custom XML
 - conversions 316
- Signed 304, 311, 312, 316
- string properties for custom XML
 - conversions 317
- Thousand 313
- time properties for custom XML
 - conversions 318
- True Output As 302
- True Value Match List 302
- Unknown Output As 303
- Use Currency Conventions 315
- Window for Two-Digit Year 307
- XML Output Form 299
- YMD Separator 306
- zoned properties for custom XML
 - conversions 320
- databases
 - accessing with XQuery 733
 - integration with Raining Data 1007
 - integration with TigerLogic XDMS 1007
- DataDirect XML Converters
 - for EDI 271
- DataDirect XQuery
 - collection() function processing 778
 - execution plans 795
 - query plans and
- date data type properties 305
- Date Format data type property 306
- datetime data type properties 308
- Date-Time Separator data type property 308
- debugging
 - cross-language debugging in XML
 - pipelines 916
 - using bookmarks in the XQuery debugger 791
 - XML pipelines 916
 - XSLT processors that support 387
- debugging stylesheets
 - example of 55
 - parser errors 503
 - processor errors 503
 - using breakpoints 494
 - what output this template generates 499
 - which template generated output 499
- debugging XQuery
 - bookmarks 791
 - displaying expressions associated with output 790
 - displaying process suspension points 790
 - displaying processing information 789
 - evaluating XPath expressions 789
 - local variables and 790
 - overview 787
 - using breakpoints 788
 - watching variables 789
- debugging XSLT
 - bookmarks and 498
 - breakpoints 494
 - determining templates associated with output 499
 - displaying instructions associated with output 498
 - displaying process suspension points 497

- displaying processing information 495
- evaluating XPath expressions 496
- example of 55
- local variables and 496
- overview 493
- watching variables 496
- decimal data type properties 308
- Decimal data type property 313
- default templates
 - description 384
 - example 31
 - how they work 354
- deploying generated code
 - for XML pipelines 924
 - for XQuery 826
 - for XSLT 408
- deploying XML pipeline code 924
- deploying XQuery code 826
- deploying XSLT code 408
- descendant axis 665
- descendant-or-self axis 668
- .dff files
 - default Stylus Studio module and 93
- Diagram** tab
 - XML Schema Editor 523
- diffing XML
 - changes that are identified 179
 - colors and symbols used by the XML Diff Viewer 182
 - command line utility for 205
 - diffing folders 183
 - diffing multiple documents 196
 - diffing two documents 194
 - merged view of diffed documents 190
 - options 201
 - overview 178
 - text view of diffed documents 189
 - tools for documents and folders 177
 - tree view of diffed documents 188
 - tuning the diffing algorithm 180
 - video demonstration 177
 - viewing documents side-by-side 188
 - when the diff is calculated 181
 - while diffing folders 187
 - XML Diff Viewer
 - adding documents 194
 - example 178
 - tool bar 191
- displaying line numbers 9
- document pane
 - display features for 231
- document wizards
 - custom document wizards 1020
 - DTD to XML 142
 - DTD to XML Schema 511
 - DTD to XML Schema (Trang) 511
 - EANCOM to XML Schema 520
 - EDIFACT to XML Schema 520
 - how to use 141
 - HTML to XML 142
 - HTML to XSLT 39
 - IATA to XML Schema 520
 - X12 to XML Schema 520
 - XML Schema to XML 141
 - XML to XML Schema 516
- document() function
 - using 705
 - XSLT Mapper and 459
- documentation
 - for XML Schema 580
 - for XQuery 810
 - Stylus Studio documentation li
- Documentation** tab
 - XML Schema Editor 525
- documents
 - creating backup copies 212
 - custom document wizards 1020
 - saving 212
- double data type properties 309
- DTD
 - code folding 146
 - converting to XML 142
 - spell checking documents 152
 - using to create XML Schema 511
- .dtd files
 - default Stylus Studio module and 93
- dynamic text
 - inserting in three-pane view 41
- dynamic Web pages
 - from static HTML 38
 - from XML
 - three-pane view 349

E

EANCOM

- creating XML Schema from 519
- creating XML Schema from EANCOM message types 520
- message types supported in Stylus Studio 271

EANCOM to XML Schema document wizard

- description 520
- options for 519

edge styles

- changing 913
- types 912

EDI

- creating custom message types 272
- creating XML Schema from 519
- creating XML Schema from EANCOM message types 520
- creating XML Schema from EDIFACT message types 520
- creating XML Schema from IATA message types 520
- creating XML Schema from X12 transaction sets 520
- dialects supported in Stylus Studio 271

EDI files

- converting to XML 271
- Custom XML Conversions module

EDI message types

- creating custom message types 272

EDIFACT

- creating XML Schema from 519
- creating XML Schema from EDIFACT message types 520
- message types supported in Stylus Studio 271

EDIFACT to XML Schema document wizard

- built-in EDI XML Converter and 283
- description 520
- options for 519
- running 520
- uses for XML Schema 283

editing XML

- bookmarks 149
- changing fonts 149
- colors used in text display 151
- commenting text 149

diffing 177

displaying line numbers 9

displaying white space in schema representations 20

features for 146

indenting text 148

inserting indents 11

line wrap 148

querying a document 12

searching text 150

Sense:X auto-completion 10

setting bookmarks 149

spell checking documents 152

tools for 146

undo 11

wrapping lines 148

editors

selecting lines in 149

Endian data type property 309, 310, 311, 312, 316

examples

bookstore.xml 640

creating dynamic Web pages from XML three-pane view 349

creating templates 358

making static Web pages dynamic 38

stylesheet 327

testing queries 643

tree representation of XML data 640

XML document structure 640

expand all 159

expanded node names

obtaining 698

extension functions

data types 379

declaring 379

finding 381

invoking 381

namespaces 380

XPath data types 380

F

False Output As data type property 302

False Value Match List data type property 302

- field names
 - specifying in Convert to XML 247
 - File Explorer
 - adding files to Projects using
 - features of 96
 - filters for 97
 - opening files with 96
 - overview 95
 - setting filters to display files in 97
 - tool bar 96
 - file systems
 - integration with Raining Data 1007
 - integration with TigerLogic XDMS 1007
 - files
 - see also* CSV, EDI, and flat files
 - adding file types to Stylus Studio 100
 - adding to projects
 - associating file types with Stylus Studio tools 94
 - converting non-XML files to XML 215
 - custom file systems 1031
 - file types and Stylus Studio module associations 93
 - making Stylus Studio the default application 100
 - opening 93
 - spell checking files in Stylus Studio 152
 - filters
 - for XQuery 649
 - Find
 - searching XML documents 157
 - fixed-width files
 - converting to XML
 - Custom XML Conversions module 215
 - flat files
 - converting to XML
 - Custom XML Conversions module 215
 - float data type properties 310
 - `floor()` function 682
 - flow ports
 - in XSLT Mapper symbols 473
 - FLWOR blocks
 - creating 771
 - Flow port 771
 - For port 770
 - illustration 770
 - in XQuery Mapper 770
 - Order by port 770
 - Return port 770
 - Where port 770
 - FLWOR expressions
 - declaring XQuery types using 748
 - examples 734
 - for 735
 - grouping 749
 - let 739
 - multiple assignments in 749
 - order by 744
 - overview 734
 - parts of 735
 - return 745
 - setting XQuery position variables using 748
 - where 743
 - FO processors
 - in XML pipelines 870
 - folders
 - diffing folder contents 183
 - Other Documents folder in Stylus Studio projects 103
 - following axis 666
 - following-sibling axis 666
 - fonts
 - changing 149
 - colors used in text display 151
 - in Convert to XML document pane 232
 - formatting objects (FO)
 - automatic tag completion 375
 - generating 394
 - function blocks
 - preserving layout in XQuery Mapper 761
 - preserving layout in XSLT Mapper 468
 - function keys
 - applying stylesheets 366
 - `function-available()` function 702
- ## G
- `generate-id()` function 705
 - generating code
 - generating Java for an XML pipeline 920
 - Go To dialog box
 - Convert to XML Editor 233

- Grid tab
 - overview 162
 - renaming nodes in 166
- grouping
 - in FLWOR expressions 749
 - XML data 950

H

- handling errors in stylesheets 503
- HMS Separator data type property 319
- Home Edition
 - description 4
- HTML
 - automatic tag completion 375
 - converting to XML 142
 - creating with XML Publisher 942
 - creating XML documents from 141
 - creating XSLT 362

I

- IATA
 - creating XML Schema from 519
 - creating XML Schema from IATA message types 520
 - message types supported in Stylus Studio 271
- IATA to XML Schema document wizard
 - description 520
 - options for 519
- id() function 692
- IDs
 - finding elements with 692
 - public IDs for XML Schemas 572
 - system IDs for XML Schemas 572
 - temporary 705
- IF blocks
 - in XQuery Mapper 776
- images
 - creating in XML Publisher reports 973
 - example of dynamic images in XSLT 45
 - including in XML Schema documentation 583
- including XML pipelines in other XML pipelines 899
- indenting tags 11

- indenting text 148
- input ports
 - in XQuery Mapper symbols 773
 - in XSLT Mapper symbols 472
- instantiating templates
 - process flow 356
- integer data type properties 311
- integration
 - with TigerLogic XDMS 1007

J

- Java 134
 - compiling code generated from XQuery 824, 922
 - compiling code generated from XSLT 406
 - configuring the JVM 134
 - defining functions in XSLT Mapper 482
 - deploying code generated from XML pipelines 924
 - deploying code generated from XQuery 826
 - deploying code generated from XSLT 408
 - dowloading Java components 135
 - extension functions for stylesheets 381
 - generating classes from a Web service 842
 - generating code from XQuery 819
 - generating code from XSLT 401
 - generating Java code for an XML pipeline 920
 - generating JAXB classes from XML Schema 585
 - Stylus Studio modules that require Java components 134
- Java Code Generation wizard for XML Pipeline
 - running 922
- Java Code Generation wizard for XQuery
 - about 819
 - compiling code from 824, 922
 - running 822
 - settings for 822
- Java Code Generation wizard for XSLT
 - about 401
 - compiling code from 406
 - running 404
 - settings for 404
- .java files
 - default Stylus Studio module and 93

JavaScript in stylesheet results 344
JAXB
 generating classes from XML Schema 585
JDK
 configuring 134
 where to download 135
JRE 134
 requirements
 for Java debugging 503
 where to download 135
JVM
 configuring 134

K

`key()` function 703
keyboard shortcuts 125

L

labeling
 XML pipelines 915
`lang()` function 679
`languages.xml` file 375
`last()` function 700
Left Padding data type property 303, 307, 315,
 317, 319
line numbers
 displaying 9
 jumping to a line in an XML document 156
line wrap 148
lists
 creating in XML Publisher reports 970
local variables
 watching during XQuery processing 790
`local-name()` function 698
logical operators
 in XSLT Mapper 479
long data type properties 312
Lookup List data type property 299

M

Mapper
 see XQuery Mapper and XSLT Mapper
mappings
 exporting as an image 456, 768
`match` attribute
 comparison with `select` attribute 333
 description 331
matched templates
 creating in XSLT Mapper 484
message types
 creating custom EDI message types 272
metrics
 XQuery performance 792
 XSLT performance 500
multidocument queries
 alternatives 706

N

`name()` function 697, 698
named templates
 creating in XSLT Mapper 484
namespace axis 668
namespaces
 extension functions 380
 obtaining information in queries 697
 stylesheets 330
`namespace-uri()` function 698
node sets
 comparing 685
`node()` function 694
nodes
 in-place editing in a schema diagram 79
 obtaining expanded names 698
 obtaining the local name 698
 obtaining the namespace URI 698
 renaming in the XML Grid tab 166
Normalize White Space data type property 317
`normalize-space()` function 674
NOT operator 677
`not()` function 679
Notes data type property 299
number data type properties 313
`number()` function 681

numbers
 converting operands to 681
 displaying line numbers in Stylus Studio editors 9

O

Omit from Output data type property 299
opening files 93
 using the File Explorer 96
operands
 converting to Booleans 678
 converting to numbers 681
 converting to strings 675
operation nodes
 labeling in XML pipelines 915
Options dialog box
 Custom Document Wizards page 1022
 Custom File Systems page 1033
 Custom Validation Engines page 1016
OR operator 677

P

Packed data type property 300
parameters in queries 701
parent axis 665
performance
 factors that affect Stylus Studio 130
 reporting XQuery metrics 792
 reporting XSLT metrics 500
performance metrics
 for XQuery 792
 for XSLT 488
pipeline
 see XML pipelines
 XML pipeline 854
position() function 689
POST data
 using POST.htm to test queries 643
post-processing XSLT 393
preceding axis 667
preceding-sibling axis 666
printing
 XML Schema 528

 XML Schema documentation 584
.prj files
 default Stylus Studio module and 93
processing-instruction() function 694
processors
 for XQuery 802, 804
 for XSLT 387, 488
profiling
 XQuery
 Profiler report 792
 stylesheet for Profiler report 792
 XSLT
 Profiler report 488
 stylesheet for Profiler report 500
projects
 adding files to
 ClearCase and 112, 114
 creating 104
 definition 101
 opening 104
 placing under source control 110
 saving 104
 setting classpaths for 108
 SourceSafe and 112
 subprojects and 104
 Visual SourceSafe and 112
 Zeus CVS and 117
properties
 XML Publisher report components 980
public IDs for XML Schemas 572
publisher
 see XML Publisher

Q

qualified names
 wildcards 699
queries
 axis syntax 663
 document element 640
 document structure 639
 function available? 702
 getting started 643
 IDs 692
 multiple documents 706

- non-XML data 632
- query language
 - attributes 648
 - Boolean expressions 677
 - comparisons 683
 - context flags 660
 - context nodes 657
 - context summary 662
 - count 700
 - filtering results 651
 - filters 649
 - getting a subscript 689
 - getting all marked-up text 644
 - `id()` function 693
 - namespaces 697
 - node names 697
 - obtaining all like-named elements 644
 - operators 684
 - path operators 660
 - quick reference for functions and methods 708
 - search context 657
 - searching by node type 694
 - selecting nodes to evaluate 656
 - subscripts 688
 - wildcards 652
 - wildcards in attributes 649
- restrictions 632
- root node 640
- sorting attributes 632
- subqueries 651
- temporary IDs 705
- tutorial 643
- variables 701
- where you can use 630
- query explain. *see* query plan
- query facility 629
- query plan
 - changing font size of 798
 - description 795
 - displaying 799
 - example 795

- how to display 799
- in Stylus Studio 795
- navigating 797
- saving as HTML 798
- toolbar 798
- tree structure 796
- query plans
 - DataDirect XQuery and 795

R

- Raining Data
 - see also* TigerLogic
 - integration with Stylus Studio 1007
- `.rdbxml` files
 - default Stylus Studio module and 93
- refactoring XML Schema nodes 81
- regular expressions
 - reference 157
 - using Search to find 157
 - using to filter output of converted documents 257
 - `xsdpattern` and 537
- relational data
 - querying with the `collection()` function 777
- removing templates 387
- Rendering data type property 301
- reports
 - creating XML reports with XML Publisher 942
 - saving an XML pipeline diagram as an image 914
- restrictions
 - queries 632
- result documents
 - getting started with 35
- Right Padding data type property 303, 307, 315, 317, 319
- root element 640
 - definition 657
- root node
 - creating matching template 358
 - default template 385
 - definition 658
 - matching template 354

- root/element default template
 - description 385
 - example 354
- round() function 683

S

- saving documents
 - automatic save 212
 - creating backup copies 212
 - options for 212
- Saxon
 - processing XQuery with 802
 - using Saxon to process XSLT 389
- Scaling Factor data type property 300, 304, 305, 309, 310, 311, 312, 314, 316, 320
- scenarios
 - definition 32
 - for Web service calls 846
 - for XQuery 800
 - performance metrics reporting 792
 - for XSLT 33
 - choosing an XSLT processor 488
 - how to clone 491
 - how to create 489
 - how to run 490
 - introduction 485
 - performance metrics reporting 488
 - setting parameter values 486
 - specifying source documents 485
- XQuery
 - validating results 806
- XSLT
 - validating results 392
- search context
 - definition 657
 - queries 657
- searching
 - text 150
 - using Find to search XML documents 157
 - using XPath to search for strings 670
- select attribute
 - comparison with match attribute 333
 - description 332
 - when there is none 354
- selecting lines of code 149
- self axis 668
- Sense:X automatic tag completion
 - description 10
 - example 10
- Sense:X tag completion
 - XSLT and 375
- short data type properties 316
- Signed data type property 304, 311, 312, 316
- SOAP requests
 - modifying 835
 - parameters for 835
- source control
 - Stylus Studio projects and 110
 - supported applications 111
- SourceSafe
 - using with Stylus Studio 112
- Spell Checker
 - personal dictionary 155
 - running 154
 - settings for 153
 - using 152
- Sruzzo
 - command line utility for running Stylus Studio 127
- string data type properties 317
- string() function 675
- string-length() function 673
- strings
 - after 672
 - before 671
 - concatenating 673
 - converting operands to 675
 - number of characters 673
 - replacing characters 674
 - searching for 670
 - substrings 672
- stylesheets
 - applied by XSLT processors 333
 - applying
 - Stylus Studio 366
 - chaining 128
 - contents 330
 - contents description 373
 - creating 362

- creating new nodes 340
 - example 327
 - for XML Schema documentation 581
 - for XQuery Profiler reports 792
 - for XSLT Profiler reports 500
 - formatting results 339
 - getting started with 27
 - introduction 327
 - namespaces 330
 - obtaining system properties 701
 - omitting source data 337
 - root element 330
 - selecting nodes for processing 335
 - Stylus Studio
 - debugging 493
 - updating
 - three-pane view 374
 - XSLT instructions 408
 - Stylus Studio
 - associating file types with 94
 - auto detect feature for Java components 136
 - benefits 345
 - building a converter URL using 288
 - command line for running 127
 - command line utilities for 127
 - debugging stylesheets 493
 - Diff tool 177
 - Home Edition description 4
 - managing performance 130
 - modules that require Java components 134
 - projects 101
 - running from the command line 127
 - using with ClearCase 112, 114
 - using with SourceSafe 112
 - using with Visual SourceSafe 112
 - using with Zeus CVS 117
 - Web services and 827
 - XML validation command line utility 130
 - XQuery command line utility for 129
 - XSLT command line utility 128
 - Stylus Studio Home Edition
 - description 4
 - StylusDiff 205
 - StylusValidator
 - command line XML validation utility 130
 - StylusXq1
 - command line XQuery utility 129
 - StylusXslt
 - command line XSLT utility 128
 - substring() function 672
 - substring-after() function 672
 - substring-before() function 671
 - sum() function 682
 - support, technical lii
 - symbols
 - element and attribute
 - in XQuery Mapper 757
 - in XSLT Mapper 464
 - lines linking nodes in XQuery Mapper 763
 - XLST function blocks
 - parts of 477
 - XLST Mapper
 - parts of 471
 - XQuery function blocks
 - parts of 773
 - XQuery Mapper
 - document 754
 - XSLT Mapper
 - document 461
 - XSLT mapper
 - XSLT instructions 471
 - syntax, notations used in this manual 1
 - system IDs for XML Schemas 572
 - system properties in stylesheets 701
 - system-property() function 701
- ## T
- tables
 - creating in XML Publisher reports 967
 - technical support lii
 - templates
 - applying 387
 - backmapping in 499
 - built-in 338
 - contents description 331
 - creating 386
 - creating named and matched templates in
 - XSLT Mapper 484
 - description of default templates 384

- displaying match patterns 29
- instantiation example 354
- introduction 330
- match attribute 333
- matched templates in XSLT Mapper 484
- matching root node 358
- more than one match 338
- named templates in XSLT Mapper 484
- no match 338
- removing 387
- rules 332
- select attribute 333
- selecting for instantiation 332
- updating 387
- viewing 382
- working with in XSLT Mapper 483
- text
 - colors used to display 151
- text blocks
 - creating in XML Publisher reports 972
- text files
 - converting to XML
 - Custom XML Conversions module 215
- text pane
 - XML Schema Editor 524
- text values
 - setting for elements and attributes in XQuery Mapper 760
 - setting for elements and attributes in XSLT Mapper
 - introduction 480
 - on the target node 481
 - using the Mapper canvas 480
- text() function 694
- text/attribute default template
 - description 385
 - example 356
- Thousand data type property 313
- three-pane view in Stylus Studio
 - result documents 35
- TigerLogic XDMS
 - connecting to collections 1008
 - integration with Stylus Studio 1007
 - processing XQuery with 803

- TigerLogicXDMS
 - creating collections in 1012
- time data type properties 318
- tool bars
 - changing appearance of Stylus Studio main tool bar 121
 - customizing Stylus Studio main tool bar 119
 - File Explorer tool bar 96
 - showing and hiding Stylus Studio main tool bar 120
 - XML Diff Viewer tool bar 191
- translate() function 674
- Tree** tab
 - XML Schema Editor 524
- True Output As data type property 302
- True Value Match List data type property 302
- tutorial for queries 643
- typographical conventions 1

U

- UDDI registries
 - searching 831
 - Web services and 831
- UN/CEFACT
 - creating XML Schema from EDIFACT message types 520
- Unknown Output As data type property 303
- unparsed-entity-uri() function 700
- updating stylesheets 374
- Use Currency Conventions data type property 315
- user-defined field names
 - display in Convert to XML Editor 230
- user-defined functions
 - in XQuery Mapper 774

V

- validating XML
 - custom validation engines for 1014
 - from the command line 130
 - standard validation engines for 1014
 - video demonstration 139

- validation engines
 - in XML pipelines 870
 - variables in queries 701
 - Video Center example
 - creating `video` template 41
 - instantiating the `video` template 43
 - making images dynamic 45
 - making summaries dynamic 46
 - making titles dynamic 44
 - video demonstrations
 - alerts for `li`
 - Convert to XML 271
 - Custom XML Conversions module 215
 - diffing XML sources 177
 - documentation alerts for `li`
 - Web Service Call Composer 827
 - XML editing and validation 139
 - XML Editor Grid tab 163
 - XML Pipeline Editor 853
 - XML Publisher 941
 - XML Schema Diagram Editor 523
 - XPath Query Editor 633
 - XQuery Mapper 717
 - viewing templates 382
 - Visual SourceSafe
 - using with Stylus Studio 112
- ## W
- Web pages
 - changing from static to dynamic HTML 38
 - creating from XML
 - three-pane view 349
 - Web Service Call Composer
 - video demonstration 827
 - Web service calls
 - specifying transport protocols for 847
 - Web services
 - creating a Web service call 828
 - generating a Java client for 842
 - invoking from an XQuery 816, 844
 - saving Web service calls 839
 - scenarios for Web service calls 846
 - SOAP requests for 835
 - testing 837
 - using in Stylus Studio 827
 - using Web service calls as XML 839
 - WSDLs and 831
 - white space
 - handling
 - XPath processor 674
 - XSL facility 340
 - tooggling display in schema representations 20
 - wildcards
 - in queries 652
 - node names 697
 - Window for Two-Digit Year data type
 - property 307
 - wizards
 - custom document wizards 1020
 - wrapping lines 148
 - `.wsc` and `.wscc` files
 - default Stylus Studio module and 93
 - WSDLs
 - displaying a WSDL document 836
 - finding WSDL URLs 831
 - searching UDDI registries for 831
- ## X
- X 12 to XML Schema document wizard
 - options for 519
 - X-12
 - creating XML Schema from 519
 - X12
 - creating XML Schema from X12 transaction sets 520
 - message types supported in Stylus Studio 271
 - X12 to XML Schema document wizard
 - description 520
 - `xln:text()` function 676
 - XML 640
 - accessing XML documents with 724
 - code folding 146
 - command line utility for validating 130
 - converting EDI to XML 271
 - converting XML to canonical XML 211
 - creating an instance from XML Schema 526
 - creating from HTML 141

- creating XML Schema from an XML document 516
- custom validation engines 1014
- diffing in Stylus Studio 177
- displaying the XML Schema associated with a document 518
- generating XML with XQuery 731
- pipeline 854
- processing using an XML pipeline 854
- spell checking documents 152
- standard validation engines for 1014
- using Web service calls as XML 839
- viewing a sample based on an XML Schema 526
- viewing an XML instance based on XML Schema 526
- XML Converters
 - building a converter URL using Stylus Studio 288
 - converting files to and from XML 215
 - URL scheme for 284
 - using in an XML pipeline 905
- XML data
 - grouping 950
- XML Diff Viewer
 - adding documents 194
 - example 178
 - merged view 190
 - options 201
 - split view 188
 - text view 189
 - tool bar 191
 - tree view 188
- XML documents
 - comparing 177
 - converting to canonical XML 211
 - creating from HTML 141
 - diffing 177
 - jumping to a line in 156
 - jumping to a matching tag 157
 - querying using XPath 211
 - root element 640
 - root node 640
 - searching 157
 - setting bookmarks in 157
 - specifying for XQuery scenarios 800
 - structure 639
 - tools for diffing 177
- XML editing
 - video demonstration 139
- XML Editor
 - displaying line numbers in 9
 - Grid tab
 - overview 162
 - renaming nodes in 166
 - jumping to a line 156
 - jumping to a matching tag 157
 - searching 157
 - setting bookmarks 157
- XML Editor Grid tab
 - video demonstration 163
- .xml files
 - default Stylus Studio module and 93
- XML instances
 - XQuery Mapper source documents and 752
 - XSLT Mapper source documents and 459
- XML Output Form data type property 299
- XML parsers
 - using in an XML pipeline 909
- XML pipeline 854
 - definition 854
 - edge styles 912
 - including XML pipelines 899
- XML pipeline canvas
 - saving as an image 914
 - zoom 912
- XML Pipeline Editor
 - video demonstration 853
- XML pipelines 916
 - annotating 911
 - converters and 905
 - debugging 916
 - deploying Java code generated from 924
 - generating Java code for 920
 - labeling 915
 - saving as an image 914
 - saving the canvas as an image 914
 - use case for 867
 - XML Converters and 905
 - XML parsers and 909
 - XML Schema and 901
 - XML serializers and 910

- XQuery and 896
- XSL-FO and 898
- XSLT and 896
- XML Publisher
 - adding data to reports 959
 - component properties 980
 - creating a list 970
 - creating a table 967
 - creating images 973
 - creating text blocks 972
 - data sources for building reports 945
 - formatting reports 984
 - how to use 943
 - overview 942
 - properties reference 999
 - report components 966
 - use case for 993
 - video demonstration 941
- XML Schema
 - code folding 146
 - converting to XML 141
 - creating 510
 - creating an XML instance from 526
 - creating from an XML document 516
 - creating from DTD 511
 - creating from EANCOM 519
 - creating from EANCOM message types 520
 - creating from EDI 519
 - creating from EDIFACT 519
 - creating from EDIFACT message types 520
 - creating from IATA 519
 - creating from IATA message types 520
 - creating from X-12 519
 - creating from X12 transaction sets 520
 - definition 510
 - detecting errors in 76
 - Diagram** tab
 - description 523
 - illustration 523
 - displaying documentation element text in the **Diagram** tab 74
 - displaying documentation using XS3P stylesheet 581
 - displaying the XML Schema associated with a document 518
 - documentation for 580
 - Documentation** tab
 - description 525
 - generating JAXB classes from 585
 - nodes
 - refactoring 81
 - printing 528
 - refactoring nodes 81
 - reference information 510
 - spell checking documents 152
 - Stylus Studio tools for 523
 - Tree** tab
 - description 524
 - using in an XML pipeline 901
 - validating 526
 - viewing an XML instance of 526
 - viewing sample XML 526
- XML Schema diagram
 - in-place editing 79
- XML Schema Diagram Editor
 - video demonstration 523
- XML Schema documentation
 - printing 584
- XML Schema Editor
 - displaying errors in text pane 76
 - text pane 524
- XML serializers
 - using in an XML pipeline 910
- XML to XML Schema document wizard 516
- XML validation
 - video demonstration 139
- XPath
 - background 630
 - benefits 631
 - choosing a version 12
 - description 630
 - evaluating expressions during XQuery processing 789
 - function blocks in XSLT Mapper
 - creating 479
 - deleting 479
 - introduction 476
 - types 477
 - mathematical function blocks in XSLT Mapper 478
 - relationship to XQuery 726

- support for 12
- XPath Query Editor 632
- XQuery and 726
- XPath expressions
 - where you can use 630
 - XML Publisher report components and 983
- XPath Query Editor
 - description 632
 - video demonstration 633
- xqDoc
 - ActiveX controls and 813
 - documentation for XQuery 810
 - viewing XQuery code samples from 814
- XQuery
 - accessing relational data with collection() functions 777
 - accessing XML documents with 724
 - accessing databases with 733
 - code folding 146
 - collection functions 777
 - command line utility for 129
 - compiling Java code generated from 824, 922
 - concat function blocks in XQuery Mapper 775
 - creating using the XQuery Mapper 750
 - creating with XML Publisher 942
 - debugging XQuery documents 787
 - declaring types in FLWOR expressions 748
 - default processor settings 804
 - deploying Java code generated from 826
 - documentation for 810
 - editor for
 - description 719
 - Mapper tab 721
 - XQuery Source tab 719
 - enabling the Profiler 793
 - evaluating XPath expressions during XQuery processing 789
 - examples 723
 - FLWOR expressions 734
 - generating for an XML Publisher report 989
 - generating Java code from 819
 - generating XML output with 731
 - grouping with 950
 - in Stylus Studio 719
 - introduction 723
 - invoking a Web service from 816, 844
 - performance metrics reporting 792
 - primer 723
 - processing with Saxon 802
 - processing with TigerLogic XDMS 803
 - processors
 - default options for 804
 - profiling 792
 - query plan 795
 - query plans 795
 - relationship to XPath 726
 - scenarios for 800
 - selecting a processor 802
 - setting a default processor 804
 - setting position variables in FLWOR expressions 748
 - specifying XML input for the XQuery 800
 - spell checking documents 152
 - using collection() functions in XQuery code 778
 - using existing XQueries in the XQuery editor 751
 - using in an XML pipeline 896
 - validating result documents 806
 - viewing source code in the Mapper 721
 - W3C definition 718
 - XPath and 726
- XQuery debugging
 - bookmarks 791
- XQuery documents
 - setting breakpoints in 788
- XQuery execution plan. *see* query plan
- .xquery files
 - default Stylus Studio module and 93
- XQuery Mapper
 - adding source documents 755
 - building a target structure 758
 - choosing source documents 752
 - condition blocks 776
 - creating FLWOR blocks 771
 - creating function blocks 772
 - creating target structure elements and attributes 759
 - creating target structure root elements 758
 - creating XQuery 750
 - document symbols 754

- element and attribute symbols in 757
 - elements and attributes
 - creating in target structures 759
 - exporting mapping as an image 768
 - FLWOR block parts 770
 - function blocks
 - about 772
 - creating 772
 - parts of 773
 - types 772
 - hiding links in 767
 - how documents are displayed 756
 - how to map nodes 763
 - IF blocks 776
 - input ports in Mapper symbols 773
 - lines linking nodes 763
 - mapping document nodes 761
 - modifying the target structure 760
 - preserving Mapper layout 761
 - removing a node from a target structure 760
 - removing node mappings 766
 - removing source documents 756
 - root elements
 - creating in target structures 758
 - setting text values 760
 - simplifying the display 767
 - source documents and XML instances 752
 - source documents for 752
 - target structures 757
 - user-defined functions 774
 - using 751
 - using FLWOR blocks 769
 - using the mouse 762
 - video demonstration 717
 - viewing source code in 721
- XQuery output
 - displaying source expressions 790
 - XQuery processing
 - call stack 790
 - displaying source expressions 790
 - displaying suspension points 790
 - selecting a processor 802
 - setting a default processor 804
 - using bookmarks 791
 - watching local variables 790
 - watching variables 789
 - XQuery processors
 - conflicts in XML pipeline 897
 - in XML pipelines 870
 - Saxon 802
 - selecting 802
 - setting a default processor 804
 - TigerLogic XDMS 803
 - XQuery Profiler
 - description 792
 - displaying the report 794
 - enabling 793
 - performance metrics captured by 793
 - report created by 792
 - XQuery scenarios
 - performance metrics reporting 792
 - XS3P stylesheet
 - display settings 583
 - displaying XML Schema documentation with 581
 - features 582
 - modifying 584
 - XSD
 - displaying documentation element text 74
 - .xsd files
 - default Stylus Studio module and 93
 - xsd:pattern
 - regular expressions and 537
 - XSL
 - additional information sources 344
 - and XSLT 630
 - definition 326
 - example 327
 - formatting objects 394
 - getting started with 325
 - inserting JavaScript in result 344
 - patterns 341
 - XSL facility
 - creating new nodes 340
 - selecting source nodes 335
 - specifying XSL patterns 341
 - white space handling 340
 - .xsl files
 - default Stylus Studio module and 93
 - XSL processor
 - applying stylesheets 333

- built-in templates 338
 - specifying result format 339
 - URI 330
- xsl:apply-imports instruction 410
- xsl:apply-templates instruction
 - comparison with xsl:for-each instruction 343
 - controlling operation order 336
 - example 336
 - more than one match 338
 - no match 338
 - no select attribute 354
 - reference 410
 - selecting nodes 335
 - specifying patterns 341
- xsl:attribute instruction 411
- xsl:attribute-set instruction 412
- xsl:call-template instruction 414
- xsl:character-map instruction 414
- xsl:choose
 - compared to xsl:if in XSLT Mapper 475
- xsl:choose instruction 417
- xsl:comment instruction 418
- xsl:copy instruction 418
- xsl:copy-of instruction 419
- xsl:debug instruction 494
- xsl:decimal-format instruction 420
- xsl:element instruction 421
- xsl:fallback instruction 422
- xsl:for-each instruction
 - comparison with xsl:apply-templates instruction 343
 - reference 422
 - selecting nodes 335
 - specifying patterns 341
- xsl:for-each-group instruction 424
- xsl:function instruction 425
- xsl:if
 - compared to xsl:choose in XSLT Mapper 475
- xsl:if instruction 426
- xsl:import instruction
 - reference 427
- xsl:import-schema instruction 427
- xsl:include instruction 429
- xsl:key instruction 430
- xsl:message instruction 431
- xsl:namespace-alias instruction 432
- xsl:number instruction 432
- xsl:otherwise instruction 433
- xsl:output instruction
 - controlling white space 341
 - reference 433
- xsl:output-character instruction 436
- xsl:param instruction 436
- xsl:preserve-space instruction 438
- xsl:processing-instruction instruction 438
- xsl:sequence instruction 439
- xsl:sort instruction 439
- xsl:strip-space instruction 441
- xsl:stylesheet instruction
 - reference 442
- xsl:template instruction
 - creating new nodes 340
 - reference 442
 - specifying patterns 341
- xsl:text instruction
 - creating white space 340
 - reference 444
- xsl:transform instruction 445
- xsl:value-of instruction
 - reference 445
 - specifying patterns 341
- xsl:variable instruction 446
- xsl:vendor property 701
- xsl:vendor-url property 701
- xsl:version property 701
- xsl:when instruction 447
- xsl:with-param instruction 447
- XSL-FO
 - creating with XML Publisher 942
 - using in an XML pipeline 898
- XSLT
 - automatic tag completion 375
 - background 326
 - chaining stylesheets 128
 - character map 415
 - choosing an XSLT processor 488
 - code folding 146
 - command line utility for 128
 - compiling Java code generated from 406
 - creating from HTML 362
 - creating using the XSLT Mapper 449

- creating with XML Publisher 942
 - debugging stylesheets 493
 - default processor settings 390
 - deploying Java code generated from 408
 - generating for an XML Publisher report 989
 - generating formatting objects 395
 - generating Java code from 401
 - grouping with 950
 - instruction blocks in XSLT Mapper 474
 - introduction 327
 - performance metrics reporting 488
 - post-processing result documents 393
 - processing with Saxon 389
 - processors
 - default options for 390
 - processors for 387
 - spell checking documents 152
 - supported versions 326
 - symbols used to represent instructions in XSLT Mapper 471
 - tags reference 408
 - using in an XML pipeline 896
 - validating result documents 392
 - viewing source code in the Mapper 451
- XSLT Editor
- displaying line numbers in 9
- XSLT Mapper
- adding instruction blocks 474
 - adding source documents 462
 - building a target structure 465
 - choosing source documents 458
 - creating target structure elements and attributes 466
 - creating target structure root elements 466
 - creating XSLT 449
 - defining Java functions in 482
 - document symbols 461
 - element and attribute symbols in 464
 - elements and attributes
 - creating in target structures 466
 - example 451
 - exporting mapping as an image 456
 - flow ports in Mapper symbols 473
 - how documents are displayed 463
 - how to map nodes 469
 - input ports in Mapper symbols 472
 - logical operators 479
 - mapping document nodes 468
 - modifying the target structure 467
 - options for 453
 - overview 450
 - overview of creating XSLT 457
 - preserving Mapper layout 468
 - processing source nodes 476
 - removing a node from a target structure 467
 - removing node mappings 470
 - removing source documents 463
 - root elements
 - creating in target structures 466
 - setting text values
 - introduction 480
 - on the target node 481
 - using the Mapper canvas 480
 - source documents and `document()` function 459
 - source documents and XML instances 459
 - source documents for 458
 - support for XSLT instructions and expressions 452
 - symbols for XSLT functions
 - parts of 477
 - symbols for XSLT instructions
 - list 471
 - parts of 471
 - target structures 465
 - using the mouse 468
 - viewing source code in 451
 - working with templates 483
- XPath function blocks
- creating 479
 - deleting 479
 - introduction 476
 - types 477
- XPath mathematical function blocks 478
- XSLT processors
- backmapping and 387
 - choosing 388
 - conflicts in XML pipeline 897
 - debugging and 387
 - in XML pipelines 870

Index

- Saxon 389
- XSLT processors supported by Stylus Studio 387
- XSLT Profiler
 - displaying the report 502
 - enabling 501
 - performance metrics captured by 501
- XSLT scenarios
 - choosing an XSLT processor 488
 - cloning 491
 - creating
 - how to 489
 - introduction 485
 - performance metrics reporting 488
 - running 490
 - setting parameter values 486
 - specifying source documents 485

Y

- YMD Separator data type property 306

Z

- Zeus CVS
 - using with Stylus Studio 117
- zoned data type properties 320