

# STYLUS STUDIO™ 6

USER GUIDE



## Stylus Studio® 6 User Guide

© 2005 Progress Software Corporation. All rights reserved.

The Progress Software Corporation products referred to in this document are also copyrighted, and all rights are reserved by Progress Software Corporation and/or its licensors, if any. 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.

Stylus Studio is a trademark of Progress Software Corporation in the U.S. and other countries.

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

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

Stylus Studio includes software developed by the Apache Software Foundation (<http://www.apache.org/>). Copyright 1999-2000 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](mailto:apache@apache.org).

Stylus Studio includes files that are subject to the Mozilla Public License Version 1.0 (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.mozilla.org/MPL/>. 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 The SAXON XSLT Processor. The Initial Developer of the Original Code is Michael Kay (<http://users.iclway.co.uk/mhkay/saxon/>). Portions created by Michael Kay are Copyright 2001. All rights reserved.

Stylus Studio includes 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.

Stylus Studio includes software developed by DecisionSoft. Copyright 2001 DecisionSoft Limited. All rights reserved.

Stylus Studio includes software developed by Thai Open Source Software Center Ltd. Copyright 2001-2003, Thai Open Source Software Center Ltd. All rights reserved.

Stylus Studio includes software developed by IBM. Copyright 1996, 1999 International Business Machines Corporation and others. All rights reserved.

Stylus Studio includes software developed by the World Wide Web Consortium. Copyright © 1994-2002 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), <http://www.w3.org/Consortium/Legal/>. All rights reserved.

Stylus Studio includes software developed by Info-ZIP. Copyright (c) 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.

Stylus Studio includes software developed by Mark Logic Corporation. Mark Logic, Mark Logic XDBC Java API and the Mark Logic logos are the trademarks of Mark Logic Corporation. Copyright 2002-2005 Mark Logic Corporation. All rights reserved.

Stylus Studio includes 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.

May 2005

---

# Contents

<b>Preface</b> .....	xxxvii
About This Manual .....	xxxvii
Conventions in This Manual .....	xxxix
Typographical Conventions .....	xxxix
Syntax Notation .....	xl
Information Alerts .....	xl
Edition Alerts .....	xl
Available Documentation .....	xli
Technical Support .....	xli
<b>Chapter 1: Getting Started with Stylus Studio®</b> .....	1
Stylus Studio Editions .....	2
Stylus Studio XML Enterprise Edition .....	2
Stylus Studio XML Professional Edition .....	3
Stylus Studio Home Edition .....	3
Edition Alerts .....	3
More Information .....	3
In This Chapter .....	4
Starting Stylus Studio .....	5
Getting Updates .....	6
Getting Help .....	6
Updating an XML Document—Getting Started .....	6
Opening a Sample XML Document .....	6
Alternatives .....	7
For more information .....	8

## Contents

---

Updating the Text of a Sample Document . . . . .	8
Displaying Line Numbers . . . . .	8
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 a Query . . . . .	14
Updating the Schema of a Sample Document . . . . .	14
Creating a Sample Schema. . . . .	15
Defining a Sample Element . . . . .	17
Adding an Element Reference to a Sample Schema . . . . .	18
Defining an Entity in a Sample Schema . . . . .	19
Exploring Other Features in a Sample Schema . . . . .	19
Updating the Tree Representation of a Sample Document . . . . .	20
Adding an Element to a Sample Document Tree. . . . .	21
Changing an Element's Data in a Sample Document Tree . . . . .	21
Adding Attributes and Other Node Types to a Sample Document Tree . . . . .	22
Adding an Entity Reference to a Sample Document Tree . . . . .	23
Updating a Sample Document Using the Grid Tab . . . . .	23
Modifying Values . . . . .	25
Moving Around the Grid . . . . .	25
Working with Stylesheets – Getting Started . . . . .	26
Opening a Sample Stylesheet . . . . .	26
XSLT Stylesheet Editor Quick Tour . . . . .	27
Parts of the XSLT Editor . . . . .	28
Exploring the XSLT Source Tab . . . . .	28
Exploring the Params/Other Tab . . . . .	31
Exploring the WYSIWYG Tab . . . . .	32
XSLT Scenarios . . . . .	33
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 <b>video</b> Template . . . . .	41
Instantiating the <b>video</b> Template . . . . .	43
Making Titles Dynamic . . . . .	44
Making Images Dynamic . . . . .	45



---

Making Summaries Dynamic . . . . .	46
Stylesheets That Generate HTML – Getting Started . . . . .	47
Video Demonstrations of the XSLT WYSIWYG Editor . . . . .	47
Descriptions . . . . .	47
System Requirements . . . . .	48
Getting Started with the XSLT WYSIWYG Editor . . . . .	48
Creating Static HTML . . . . .	49
Defining Dynamic Contents . . . . .	52
Adding a Table with Dynamic Contents . . . . .	53
Using the Properties Window . . . . .	55
Making a Static Web Page Dynamic Using the WYSIWYG Editor . . . . .	58
Before You Begin . . . . .	59
Making Repeating Table Rows in the WYSIWYG Editor . . . . .	59
Making Contents Dynamic in the WYSIWYG Editor . . . . .	60
Rendering Images as Dynamic Content in the WYSIWYG Editor . . . . .	61
Using Conditional If Processing in the WYSIWYG Editor . . . . .	63
Using the XSLT Mapper – Getting Started . . . . .	64
Opening the XSLT Mapper . . . . .	64
Mapping Nodes in Sample Files . . . . .	66
Saving the Stylesheet and Previewing the Result . . . . .	69
Deleting Links in Sample Files . . . . .	70
Defining Additional Processing in Sample Files . . . . .	71
Debugging Stylesheets – Getting Started . . . . .	71
Setting Up Stylus Studio to Debug Sample Files . . . . .	72
Inserting a Breakpoint in the Sample Stylesheet . . . . .	73
Gathering Debug Information About the Sample Files . . . . .	75
The Variables Window . . . . .	76
The Call Stack Window . . . . .	76
The Watch Window . . . . .	77
Ending Processing During a Debug Session . . . . .	78
Defining a DTD – Getting Started . . . . .	80
Process Overview . . . . .	80
Creating a Sample DTD . . . . .	80
Defining Data Elements in a Sample DTD . . . . .	81
Defining the Container Element in a Sample DTD . . . . .	82
Defining Structure Rules in a Sample DTD . . . . .	82
Examining the Tree of a Sample DTD . . . . .	84

## Contents

---

Defining an XML Schema Using the Diagram Tab – Getting Started . . . . .	85
Introduction to the Diagram Tab . . . . .	86
Diagram Pane . . . . .	87
Text Pane . . . . .	92
Definition Browser. . . . .	94
Editing Tools of the Diagram Tab . . . . .	95
Menus and Tool Bars . . . . .	95
In-place Editing . . . . .	95
Drag-and-Drop . . . . .	96
QuickEdit . . . . .	97
Refactoring . . . . .	97
Description of Sample XML Schema. . . . .	100
Defining a complexType in a Sample XML Schema in the Diagram View . . . . .	100
Defining the Name of a Sample complexType in the Diagram View . . . . .	101
Adding an Attribute to a Sample complexType in the Diagram View . . . . .	102
Adding Elements to a Sample complexType in the Diagram View . . . . .	103
Adding Optional Elements to a Sample complexType in the Diagram View . . . . .	104
Adding an Element That Contains Subelements to a complexType in the Diagram View . . . . .	104
Choosing the Element to Include in a Sample complexType in the Diagram View . . . . .	106
Defining Elements of the Sample complexType in the Diagram View . . . . .	108
Opening Files in Stylus Studio . . . . .	110
Types of Files Recognized by Stylus Studio . . . . .	110
Opening Unknown File Types . . . . .	111
Opening Files Stored on Berkeley DB XML. . . . .	111
Modifications to Open Files. . . . .	112
Using the File Explorer. . . . .	112
How to Use the File Explorer to Open Files . . . . .	113
Other Features of the File Explorer . . . . .	113
Working with the File Explorer Filter . . . . .	114
Dragging and Dropping Files in the Stylus Studio. . . . .	115
Other Ways to Open Files in Stylus Studio . . . . .	116
Adding File Types to Stylus Studio . . . . .	117
Deleting File Types . . . . .	118
Working with Projects. . . . .	118
Displaying the Project Window . . . . .	119
Displaying Path Names . . . . .	120
Other Documents . . . . .	120

---

Creating Projects and Subprojects . . . . .	121
Saving Projects . . . . .	121
Opening Projects . . . . .	121
Recently Opened Projects . . . . .	122
Adding Files to Projects . . . . .	122
Other Ways to Add Files to Projects . . . . .	123
Copying Projects . . . . .	123
Rearranging the Files in a Project . . . . .	124
Removing Files from Projects . . . . .	124
Closing and Deleting Projects . . . . .	124
Closing . . . . .	124
Deleting . . . . .	125
Setting a Project Classpath . . . . .	125
Specifying Multiple Classpaths . . . . .	125
How to Set a Project Classpath . . . . .	125
Using Stylus Studio with Source Control Applications . . . . .	127
Tested Source Control Applications . . . . .	128
Prerequisites . . . . .	128
Recursive Selection . . . . .	128
Using Stylus Studio with Microsoft Visual SourceSafe . . . . .	129
Using Stylus Studio with ClearCase . . . . .	131
Using Stylus Studio with Zeus CVS . . . . .	134
Specifying Advanced Source Control Properties . . . . .	135
Customizing Tool Bars . . . . .	136
Tool Bar Groups . . . . .	136
Showing/Hiding Tool Bar Groups . . . . .	137
Changing Tool Bar Appearance . . . . .	138
Specifying Stylus Studio Options . . . . .	138
Modifying Java Options . . . . .	139
About Java Virtual Machine Options . . . . .	139
About Java Compiler Options . . . . .	140
About External JVM Options . . . . .	141
How to Modify Java Settings . . . . .	142
Setting Module Options . . . . .	142
XML Diff . . . . .	142
XML Editor . . . . .	142
XSLT Editor . . . . .	143
Java . . . . .	144
Defining Custom Tools . . . . .	144

## Contents

---

Defining Keyboard Shortcuts . . . . .	145
How to Define a Keyboard Shortcut . . . . .	145
Deleting a Keyboard Shortcut . . . . .	147
Using Stylus Studio from the Command Line . . . . .	147
Invoking Stylus Studio from the Command Line. . . . .	148
Applying a Stylesheet from the Command Line . . . . .	148
Executing an XQuery from the Command Line. . . . .	149
Validating XML from the Command Line. . . . .	150
Managing Stylus Studio Performance. . . . .	151
Troubleshooting Performance . . . . .	151
Changing the Schema Refresh Interval . . . . .	152
Checking for Modified Files. . . . .	153
Changing the Recursion Level or Allocated Stack Size. . . . .	153
Automatically Opening the Last Open Files . . . . .	154
<b>Chapter 2: Editing and Querying XML . . . . .</b>	<b>155</b>
Creating XML Documents . . . . .	156
Other Ways to Create XML . . . . .	156
Converting Text Files to XML Documents . . . . .	156
Alternative to Document Wizards . . . . .	157
About CSV File Contents. . . . .	157
Delimiting Commas . . . . .	157
Structure of Resulting XML Documents . . . . .	158
About the Default Values . . . . .	159
Running the Convert CSV to XML and Convert Fixed-Width to XML Document Wizards .	159
Specifying User-Defined Columns . . . . .	161
Converting HTML to XML Documents . . . . .	163
Updating XML Documents. . . . .	163
Choosing a View. . . . .	164
For More Information . . . . .	164
Saving Your Work . . . . .	164
Ensuring Well-Formedness . . . . .	165
Reverting to Saved Version . . . . .	165
Updating Java Server Pages as XML Documents . . . . .	165
Updating XML Documents Using the Text Editor . . . . .	166
Common Text Editing Functions and Tools. . . . .	166
Sense:X . . . . .	167
Indent . . . . .	167
Line Wrap. . . . .	167

---

Spell Checking . . . . .	168
Font . . . . .	168
Comments . . . . .	169
Bookmarks . . . . .	169
Search . . . . .	169
Use of Colors in the Text Tab . . . . .	169
How to Change Text Colors . . . . .	170
Using the Spell Checker . . . . .	171
Default Spell Checking . . . . .	171
Manual Spell Checking . . . . .	171
Specifying Spell Checker Settings . . . . .	172
How to Spell Check a Document . . . . .	173
Using the Personal Dictionary . . . . .	174
Updating DOM Tree Structures . . . . .	176
Displaying All Nodes in the Tree View . . . . .	177
Adding a Node in the Tree View . . . . .	177
Deleting a Node in the Tree View . . . . .	178
Moving a Node in the Tree View . . . . .	178
Changing the Name or Value of a Node in the Tree View . . . . .	178
Obtaining the XPath for a Node . . . . .	179
Updating XML Documents Using the Grid Tab . . . . .	179
Features of the Grid Tab . . . . .	181
Layout of the Grid Tab . . . . .	181
Expanding and Collapsing Nodes . . . . .	182
Collapsing Empty Nodes . . . . .	182
Renaming Nodes . . . . .	184
Resizing Columns . . . . .	184
Showing Row Tag Names . . . . .	184
Moving Around the Grid Tab . . . . .	185
Selecting Items in the Grid . . . . .	186
How Grid Changes Affect the XML Document . . . . .	186
Types of Changes that Affect the Document . . . . .	187
Working with Rows . . . . .	187
Reordering Rows . . . . .	188
Adding and Deleting Rows . . . . .	188
Working with Columns . . . . .	189
Selecting a Column . . . . .	189
Adding Columns . . . . .	189
Deleting Columns . . . . .	190

## Contents

---

Reordering Columns .....	190
Renaming Columns .....	191
Changing a Value .....	191
Working with Tables .....	191
Adding a Nested Table .....	192
Moving a Nested Table .....	193
Deleting a Table .....	193
Sorting a Table .....	194
Copying a Table as Tab-Delimited Text .....	194
Diffing Folders and XML Documents .....	195
Overview .....	196
Sources and Targets .....	197
The Diff Configuration File .....	197
What Diffs Are Calculated? .....	197
Tuning the Diffing Algorithm .....	198
When Does the Diff Run? .....	199
Running the Diff Manually .....	200
Symbols and Background Colors .....	200
Diffing Folders .....	201
Features .....	202
How to Diff Folders .....	203
How to Diff Documents from the Diff Folders Dialog Box .....	205
The XML Diff Viewer .....	205
Split View - Tree .....	206
Split View - Text .....	207
Merged View .....	208
View Symbols and Colors .....	209
The XML Diff Viewer Tool Bar .....	209
Tools for Working with Documents .....	212
Removing a Target Document .....	213
Diffing a Pair of XML Documents .....	213
How to Diff a Pair of Documents .....	213
Diffing Multiple Documents .....	214
Document Focus .....	214
Symbols Used in the Target Document Window .....	215
How to Diff Multiple Documents .....	218
Modifying Default Diff Settings .....	219
Opening the Options Dialog Box .....	220
Engine Settings .....	221

Presentation Options . . . . .	223
Running the Diff Tool from the Command Line . . . . .	223
Restrictions . . . . .	224
Usage . . . . .	224
Using Schemas with XML Documents . . . . .	226
Associating an External Schema With a Document . . . . .	226
Having Stylus Studio Generate a Schema . . . . .	227
Validating XML Documents . . . . .	227
Updating a Document's Schema . . . . .	228
Removing the Association Between a Document and a Schema . . . . .	228
Converting XML to Its Canonical Form . . . . .	229
Querying XML Documents Using XPath . . . . .	229
Steps for Querying a Document . . . . .	229
Displaying Query Results . . . . .	230
Saving Query Results . . . . .	230
Moving Around in XML Documents . . . . .	231
Line Numbers . . . . .	231
Bookmarks . . . . .	231
Tags . . . . .	232
Find . . . . .	232
Learning About Regular Expressions . . . . .	232
Printing XML Documents . . . . .	232
Saving XML Documents . . . . .	233
Options for Saving Documents . . . . .	233
More About Backup Files . . . . .	233
Opening a Backup File . . . . .	234
<b>Chapter 3: Converting Non-XML Files to XML . . . . .</b>	<b>235</b>
Overview of Convert to XML . . . . .	236
File Support . . . . .	237
Using Convert to XML . . . . .	238
Other Ways to Convert Files to XML . . . . .	238
Choosing an Input File . . . . .	239
The Convert to XML Editor . . . . .	240
Document Pane . . . . .	241
Example – .txt Files . . . . .	241
Display of Delimiting and Control Characters . . . . .	242
Field Names . . . . .	243
Document Pane Display Features . . . . .	244

## Contents

---

Moving Around the Document .....	246
Properties Window .....	248
How Properties are Organized .....	249
Properties for Fixed-Width and Line-Oriented Input Files .....	249
Schema Pane .....	250
Parts of an Input File .....	251
Regions .....	251
Region Types .....	252
Managing Regions .....	252
Rows .....	252
Fields .....	253
Component and Sub-Component Fields .....	253
Working with Regions .....	253
Converting the Region Type .....	253
How to Convert a Region Type .....	255
Adjusting Fixed-Width Regions .....	256
Example .....	256
Defining and Joining Regions .....	257
Defining a Region .....	257
Joining Regions .....	258
Controlling Region Output .....	259
Working with Fields .....	259
Naming Fields .....	260
Using the Element Name Source Property .....	260
More About Using Rows for Field Names .....	262
How to Name Fields .....	262
Defining Fields .....	263
Creating Notes for Fields .....	265
Component and Sub-Component Fields .....	266
Controlling XML Output .....	267
Specifying Element Names .....	268
Specifying Format .....	269
Omitting Regions and Fields, and Rows .....	269
Pattern Matching .....	270
Example .....	270
Sample Regular Expressions .....	271
Specifying Multiple Match Patterns .....	272
Working with Nodes .....	272



---

Using Lookup Lists . . . . .	275
Defining Lookup Lists . . . . .	276
Working with Lookup Lists . . . . .	277
Using Key=Value Characters . . . . .	277
Creating an Adapter . . . . .	278
Specifying File Settings. . . . .	279
How to Create an Adapter. . . . .	279
Using Adapters in Stylus Studio . . . . .	280
Built-In Adapters. . . . .	281
How to Open a File Using an Adapter . . . . .	281
Using the File Explorer. . . . .	281
Using the Open Dialog Box . . . . .	281
More About Converting EDI. . . . .	284
Using Convert to XML for Converting EDI . . . . .	284
Using the Built-In EDI Adapter . . . . .	284
Where to Find It . . . . .	286
Validating XML from/to EDI . . . . .	287
Invoking an Adapter Programmatically. . . . .	287
Adapter URLs . . . . .	287
Where Adapter URLs are Displayed in Stylus Studio . . . . .	289
The StylusFile Object . . . . .	290
Constructing Your Own Adapter URL . . . . .	291
Using the URL in the Select XML Converter Dialog Box. . . . .	291
Using the URL in the Properties Window . . . . .	292
Example – demo.bat . . . . .	293
Demonstration Files . . . . .	293
demo.java . . . . .	294
More About the Stylus Studio File System Java API . . . . .	298
Javadoc . . . . .	298
User-Defined Adapter Properties Reference . . . . .	298
Input File Properties . . . . .	299
XML Output File Properties . . . . .	300
Region Properties . . . . .	301
Row Properties . . . . .	304
Field Properties . . . . .	305
Type-Specific Properties. . . . .	307
Specifying Control Characters . . . . .	311

<b>Chapter 4: Working with XSLT</b> .....	315
Getting Started with XSLT .....	315
What Is XSLT? .....	316
What Is a Stylesheet? .....	317
Example of a Stylesheet .....	317
About Stylesheet Contents .....	320
What Is a Template? .....	320
Contents of a Template .....	321
Determining Which Template to Instantiate .....	322
How the <b>select</b> and <b>match</b> Attributes Are Different .....	323
How the XSLT Processor Applies a Stylesheet .....	323
Instantiating the First Template .....	324
Selecting Source Nodes to Operate On .....	325
Controlling the Order of Operation .....	326
Omitting Source Data from the Result Document .....	327
When More Than One Template Is a Match .....	328
When No Templates Match .....	328
Controlling the Contents of the Result Document .....	329
Specifying Result Formatting .....	329
Creating New Nodes in the Result Document .....	330
Controlling White Space in the Result .....	330
Specifying XSLT Patterns and Expressions .....	331
Examples of Patterns and Expressions .....	331
Frequently Asked Questions About XSLT .....	333
Sources for Additional XSLT Information .....	334
Benefits of Using Stylus Studio .....	335
Structural Data View .....	335
Sophisticated Editing Environment .....	336
XSLT and Java Debugging Features .....	337
Integrated XML Parser/XSLT Processor .....	339
Tutorial: Understanding How Templates Work .....	339
Creating a New Sample Stylesheet .....	340
Understanding How the Default Templates Work .....	344
Instantiating the Template That Matches the Root Node .....	344
Instantiating the Root/Element Default Template .....	345
Instantiating the Text/Attribute Default Template .....	346
Illustration of Template Instantiations .....	347
Editing the Template That Matches the Root Node .....	348
Creating a Template That Matches the <b>book</b> Element .....	349

---

Creating a Template That Matches the <b>author</b> Element . . . . .	350
Working with Stylesheets . . . . .	351
About the XSLT Editor . . . . .	352
Creating Stylesheets . . . . .	353
Creating a Stylesheet from HTML . . . . .	354
Specifying Stylesheet Parameters and Options . . . . .	354
Applying Stylesheets . . . . .	357
About Applying Stylesheets . . . . .	357
Results of Applying a Stylesheet . . . . .	358
Applying Stylesheets to Large Data Sets . . . . .	360
Creating a Scenario . . . . .	360
Cloning Scenarios . . . . .	362
Saving Scenario Meta-Information . . . . .	362
Applying a Stylesheet to Multiple Documents . . . . .	363
Applying the Same Stylesheet in Separate Operations . . . . .	363
Applying a Stylesheet to Multiple Documents in One Operation . . . . .	363
About Stylesheet Contents . . . . .	364
Contents Provided by Stylus Studio . . . . .	364
Contents You Can Add . . . . .	365
Updating Stylesheets . . . . .	365
Dragging and Dropping from Schema Tree into XSLT Editor . . . . .	365
Using Sense:X Automatic Tag Completion . . . . .	366
Using Sense:X to Ensure Well-Formed XML . . . . .	366
Using Standard Editing Tools . . . . .	367
Saving Stylesheets . . . . .	367
Using Updated Stylesheets . . . . .	368
Creating Stylesheets That Generate HTML . . . . .	369
Descriptions of WYSIWYG Terms . . . . .	370
Inserting Contents in the HTML Editor . . . . .	371
Displaying a Repeating Element in the HTML Editor . . . . .	372
Adding Conditional Processing in the HTML Editor . . . . .	372
Specifying Choose Conditional Processing in the HTML Editor . . . . .	373
Specifying If Conditional Processing in the HTML Editor . . . . .	374
Instantiating Templates in the HTML Editor . . . . .	375
Calling a Named Template . . . . .	375
Specifying Properties and Attributes in the HTML Editor . . . . .	376
Specifying Extension Functions in Stylesheets . . . . .	376
Using an Extension Function in Stylus Studio . . . . .	377
Basic Data Types . . . . .	378

## Contents

---

Declaring an XSLT Extension Function . . . . .	378
Working with XPath Data Types . . . . .	379
Declaring an Extension Function Namespace . . . . .	379
Invoking Extension Functions . . . . .	380
Finding Classes and Finding Java. . . . .	380
Debugging Stylesheets That Contain Extension Functions . . . . .	380
Working with Templates. . . . .	381
Viewing Templates. . . . .	381
Viewing a List of Templates . . . . .	382
Viewing a Specific Template . . . . .	382
Checking if a Template Generates Output . . . . .	383
Using Stylus Studio Default Templates . . . . .	383
Contents of a New Stylesheet Created by Stylus Studio . . . . .	383
About the Root/Element Built-In Template . . . . .	384
About the Text/Attribute Built-In Template . . . . .	384
Creating Templates. . . . .	385
Saving a Template . . . . .	385
Applying Templates . . . . .	386
Updating Templates . . . . .	386
Deleting Templates. . . . .	386
Using an External XSLT Processor . . . . .	386
How to Use an External Processor . . . . .	387
Passing Parameters. . . . .	388
Setting Default Options for Processors. . . . .	388
Validating Result Documents . . . . .	389
Post-processing Result Documents . . . . .	391
Generating Formatting Objects. . . . .	392
Developing Stylesheets That Generate FO. . . . .	393
Troubleshooting FOP Errors. . . . .	393
Viewing the FO Sample Application . . . . .	394
Deploying Stylesheets That Generate FO. . . . .	396
Example . . . . .	396
Using Apache FOP to Generate NonPDF Output . . . . .	397
Generating Scalable Vector Graphics . . . . .	398
About SVG Viewers. . . . .	398
Running the SVG Example . . . . .	398
Generating Java Code for XSLT. . . . .	399
Scenario Settings . . . . .	399
Choosing Scenarios . . . . .	400

---

Java Code Generation Settings . . . . .	401
How to Generate Java Code for XSLT . . . . .	402
Compiling Generated Code . . . . .	403
How to Modify the Stylus Studio Classpath . . . . .	403
How to Compile and Run Java Code in Stylus Studio . . . . .	404
XSLT Instructions Quick Reference . . . . .	405
xsl:apply-imports . . . . .	406
xsl:apply-templates . . . . .	406
Format . . . . .	406
Description . . . . .	406
Example . . . . .	407
xsl:attribute . . . . .	407
Format . . . . .	407
Description . . . . .	408
Example . . . . .	408
xsl:attribute-set . . . . .	409
Format . . . . .	409
Description . . . . .	409
Example . . . . .	410
xsl:call-template . . . . .	411
Format . . . . .	411
Description . . . . .	411
xsl:choose . . . . .	411
Format . . . . .	411
Description . . . . .	412
xsl:comment . . . . .	412
Format . . . . .	412
Description . . . . .	412
Example . . . . .	413
xsl:copy . . . . .	413
Format . . . . .	413
Description . . . . .	413
Example . . . . .	413
xsl:copy-of . . . . .	414
Format . . . . .	414
Description . . . . .	414
xsl:decimal-format . . . . .	414
Format . . . . .	414
Description . . . . .	414

## Contents

---

xsl:element . . . . .	416
Format . . . . .	416
Description . . . . .	416
Example . . . . .	416
xsl:fallback . . . . .	417
xsl:for-each . . . . .	417
Format . . . . .	417
Description . . . . .	417
Example . . . . .	418
xsl:if . . . . .	419
Format . . . . .	419
Description . . . . .	419
Example . . . . .	419
xsl:import . . . . .	420
Format . . . . .	420
Description . . . . .	420
xsl:include. . . . .	420
Format . . . . .	420
Description . . . . .	421
xsl:key. . . . .	421
Format . . . . .	421
Description . . . . .	421
xsl:message. . . . .	422
Format . . . . .	422
Description . . . . .	422
xsl:namespace-alias . . . . .	423
Format . . . . .	423
Description . . . . .	423
xsl:number . . . . .	423
Format . . . . .	423
Description . . . . .	423
Example . . . . .	424
xsl:otherwise. . . . .	425
xsl:output . . . . .	425
Format . . . . .	425
Description . . . . .	425
xsl:param. . . . .	427
Format . . . . .	427
Description . . . . .	427

xsl:preserve-space . . . . .	428
xsl:processing-instruction . . . . .	428
Format . . . . .	428
Description . . . . .	428
Example . . . . .	429
xsl:sort . . . . .	429
Format . . . . .	429
Description . . . . .	429
Example . . . . .	431
xsl:strip-space . . . . .	431
xsl:stylesheet . . . . .	432
Format . . . . .	432
Description . . . . .	432
xsl:template . . . . .	432
Format . . . . .	432
Description . . . . .	432
xsl:text . . . . .	434
Format . . . . .	434
Description . . . . .	434
Examples . . . . .	434
xsl:transform . . . . .	435
xsl:value-of . . . . .	435
Format . . . . .	435
Description . . . . .	435
Example . . . . .	435
xsl:variable . . . . .	436
Format . . . . .	436
Description . . . . .	436
xsl:when . . . . .	437
xsl:with-param . . . . .	437
Format . . . . .	437
Description . . . . .	437
Example . . . . .	438

**Chapter 5: Creating XSLT Using the XSLT Mapper . . . . . 439**

Overview of the XSLT Mapper . . . . .	439
Example . . . . .	441
Graphical Support for Common XSLT Instructions and Expressions . . . . .	442
Setting Options for the XSLT Mapper . . . . .	443

## Contents

---

Ensuring That Stylesheets Output Valid XML . . . . .	444
Steps for Mapping XML to XML . . . . .	444
Source Documents . . . . .	445
Choosing Source Documents . . . . .	445
Source Documents and XML Instances . . . . .	446
Types of associations . . . . .	446
Source document icons . . . . .	448
How to change a source document association . . . . .	448
How to Add a Source Document . . . . .	448
How to Remove a Source Document . . . . .	450
How Source Documents are Displayed . . . . .	450
Document structure symbols . . . . .	451
Getting source document details . . . . .	451
Target Structures . . . . .	451
Using an Existing Document . . . . .	452
Building a Target Structure . . . . .	452
Modifying the Target Structure . . . . .	453
Adding a Node . . . . .	453
Removing a Node . . . . .	454
Mapping Source and Target Document Nodes . . . . .	454
Preserving Mapper Layout . . . . .	454
Left and Right Mouse Buttons Explained . . . . .	455
How to Map Nodes . . . . .	456
Removing Source-Target Maps . . . . .	456
Working with XSLT Instructions in XSLT Mapper . . . . .	456
What XSLT Instructions Are Represented Graphically . . . . .	457
Instruction Block Ports . . . . .	458
Specifying Values for Ports . . . . .	458
Understanding Input Ports . . . . .	459
Specifying Values for Input Ports . . . . .	459
Red Input Ports . . . . .	460
The Flow Port . . . . .	460
Adding an Instruction Block to the XSLT Mapper . . . . .	460
Notes About Creating Instruction Blocks . . . . .	461
xsl:if and xsl:choose . . . . .	462
Editing xsl:choose Instruction Properties . . . . .	463
Processing Source Nodes . . . . .	464
XPath Function Blocks . . . . .	464
Parts of a Function Block . . . . .	464



---

Types of Function Blocks . . . . .	465
XPath Mathematical Functions . . . . .	466
Creating a Function Block . . . . .	466
Deleting a Function Block . . . . .	466
Logical Operators . . . . .	467
Setting a Text Value . . . . .	467
Example . . . . .	467
How to Set a Text Value on the Mapper Canvas . . . . .	468
How to Set a Text Value on the Target Node . . . . .	469
Defining Java Functions in the XSLT Mapper . . . . .	469
About Adding Java Class Files . . . . .	470
Creating and Working with Templates . . . . .	470
What Happens When You Create a Template . . . . .	470
How to Create a Named or Matched Template . . . . .	471
Creating an XSLT Scenario . . . . .	472
Overview of Scenario Features . . . . .	473
XML Source Documents . . . . .	473
Global Parameters . . . . .	474
XSLT Processors . . . . .	475
Performance Metrics Reporting . . . . .	475
Result Document Validation . . . . .	476
Post-Processing Result Documents . . . . .	476
How to Create a Scenario . . . . .	476
How to Run a Scenario . . . . .	477
How to Clone a Scenario . . . . .	478
<b>Chapter 6: Debugging Stylesheets . . . . .</b>	<b>479</b>
Steps for Debugging Stylesheets . . . . .	480
Using Breakpoints . . . . .	480
Inserting Breakpoints . . . . .	480
Removing Breakpoints . . . . .	481
Start Debugging . . . . .	481
Viewing Processing Information . . . . .	481
Watching Particular Variables . . . . .	482
Evaluating XPath Expressions in the Current Processor Context . . . . .	482
Obtaining Information About Local Variables . . . . .	482
Determining the Current Context in the Source Document . . . . .	483
Displaying a List of Process Suspension Points . . . . .	483
Displaying XSLT Instructions for Particular Output . . . . .	484

## Contents

---

Using Bookmarks . . . . .	484
Determining Which Template Generated Particular Output . . . . .	485
Determining the Output Generated by a Particular Template . . . . .	486
Profiling XSLT Stylesheets . . . . .	486
Enabling the Profiler . . . . .	488
Displaying the XSLT Profiler Report . . . . .	489
Handling Parser and Processor Errors . . . . .	489
Debugging Java Files . . . . .	489
Requirements for Java Debugging . . . . .	490
Setting Options for Debugging Java . . . . .	491
Using the Java Editor . . . . .	491
Stylus Studio and JVM . . . . .	492
Example of Debugging Java Files . . . . .	493
Setting Up to Debug Sample Java/XSLT Application . . . . .	493
Inserting a Breakpoint in the Sample Java/XSLT Application . . . . .	494
Gathering Debug Information About the Sample Java/XSLT Application . . . . .	494
<b>Chapter 7: Defining XML Schemas . . . . .</b>	<b>497</b>
What Is an XML Schema? . . . . .	498
Reference Information . . . . .	498
Creating an XML Schema in Stylus Studio . . . . .	498
Creating Your Own XML Schema . . . . .	499
Creating XML Schema from a DTD . . . . .	499
Using the DTD to XML Schema Document Wizard . . . . .	499
Using the DTD to XML (Trang) Document Wizard . . . . .	500
Creating XML Schema from an XML Document . . . . .	504
Using the XML to XML Schema Document Wizard . . . . .	504
Using the Create Schema from XML Content Feature . . . . .	505
Displaying the New XML Schema . . . . .	506
Creating XML Schema from EDIFACT Messages . . . . .	506
Setting Wizard Options . . . . .	506
Running the EDIFACT to XML Schema Documentation Wizard . . . . .	506
Working with XML Schema in Stylus Studio . . . . .	507
Views in the XML Schema Editor . . . . .	508
Validating XML Schema . . . . .	511
Updating XML Schema Associated with a Document . . . . .	511
Viewing Sample XML . . . . .	512
Using XML Schema in XQuery and XSLT Mapper . . . . .	513

---

Printing . . . . .	513
Printing XML Schema . . . . .	513
Printing XML Schema Documentation . . . . .	513
Node Properties . . . . .	513
Working with Properties in the Diagram . . . . .	514
Getting Started with XML Schema in the Tree View . . . . .	514
Description of Sample XML Schema . . . . .	515
Tips for Adding Nodes . . . . .	515
Defining a complexType in a Sample XML Schema in the Tree View . . . . .	515
Defining the Name of the Sample complexType in the Tree View . . . . .	516
Adding an Attribute to a Sample complexType in the Tree View . . . . .	516
Adding Elements to a Sample complexType in the Tree View . . . . .	517
Adding Optional Elements to a Sample complexType in the Tree View. . . . .	517
Adding an Element That Contains Subelements to a complexType in the Tree View. . . . .	518
Choosing the Element to Include in the Sample complexType in the Tree View . . . . .	519
Defining Elements of the Sample complexType in the Tree View. . . . .	520
Defining simpleTypes in XML Schemas. . . . .	520
About simpleTypes in XML Schemas . . . . .	521
Examples of simpleTypes in an XML Schema. . . . .	521
Defining a simpleType in the Diagram View . . . . .	522
Before You Begin . . . . .	523
Defining an Atomic simpleType. . . . .	523
Specifying a Restriction for a simpleType – QuickEdit . . . . .	523
Specifying a Restriction for a simpleType – Manually . . . . .	525
Defining List and Union simpleTypes . . . . .	526
Defining a simpleType in the Tree View . . . . .	527
About Facet Types for simpleTypes . . . . .	528
Defining List and Union simpleTypes in the Tree View . . . . .	530
Defining complexTypes in XML Schemas . . . . .	530
Defining complexTypes That Contain Elements and Attributes – Diagram View. . . . .	531
Adding Nodes to a complexType . . . . .	532
Choosing an Element . . . . .	532
Including All Elements . . . . .	533
Specifying the Sequence of Elements. . . . .	534
Reordering Nodes . . . . .	534
Combining the Sequence and Choice Modifiers . . . . .	535
Defining complexTypes That Contain Elements and Attributes – Tree View . . . . .	535
Defining complexTypes That Mix Data and Elements. . . . .	537
Diagram View. . . . .	537

## Contents

---

Tree View . . . . .	538
Defining complexTypes That Contain Only Attributes . . . . .	539
Diagram View . . . . .	539
Tree View . . . . .	540
Defining Elements and Attributes in XML Schemas . . . . .	540
Defining Elements That Carry Attributes and Contain Data in XML Schemas. . . . .	541
Diagram View . . . . .	541
Tree View . . . . .	543
Defining Elements That Contain Subelements in XML Schemas . . . . .	544
Diagram View . . . . .	544
Tree View . . . . .	545
Adding an Identity Constraint to an Element . . . . .	545
Example of an Identity Constraint . . . . .	546
Diagram View . . . . .	547
Tree View . . . . .	548
Defining Groups of Elements and Attributes in XML Schemas. . . . .	549
Defining Groups of Elements in XML Schemas – Diagram View . . . . .	549
Alternative . . . . .	550
Defining Groups of Elements in XML Schemas – Tree View . . . . .	550
Defining attributeGroups in XML Schemas – Diagram View . . . . .	551
Defining attributeGroups in XML Schemas – Tree View . . . . .	552
Adding Comments, Annotation, and Documentation Nodes to XML Schemas. . . . .	553
Comments . . . . .	553
Annotations. . . . .	553
Diagram View . . . . .	553
Tree View . . . . .	554
Moving a Comment or Annotation . . . . .	554
Example . . . . .	555
Defining Notations . . . . .	556
Diagram View . . . . .	556
Tree View . . . . .	556
Referencing External XML Schemas . . . . .	557
Ways to Reference XML Schemas. . . . .	557
Including an XML Schema . . . . .	557
Importing an XML Schema . . . . .	558
Redefining an XML Schema . . . . .	558
Where You Can Reference XML Schemas . . . . .	558
What to Do Next . . . . .	559
Referencing XML Schemas in the Diagram View . . . . .	559

---

Referencing XML Schemas in the Tree View . . . . .	561
Redefining Nodes . . . . .	561
Extensions and Restrictions . . . . .	561
Specifying Restriction Facets . . . . .	562
How to Redefine a Node . . . . .	562
Generating Documentation for XML Schema . . . . .	564
XS3P Stylesheet Overview . . . . .	565
XS3P Stylesheet Features . . . . .	566
XS3P Stylesheet Settings . . . . .	567
Modifying the XS3P Stylesheet . . . . .	568
XSDdoc Stylesheet Overview . . . . .	569
XSDdoc Stylesheet Settings . . . . .	569
Choosing a Display Option for XML Schema Documentation . . . . .	570
Saving XML Schema Documentation . . . . .	571
Printing XML Schema Documentation . . . . .	571
Generating JAXB Classes . . . . .	571
What Stylus Studio Generates . . . . .	572
How to Generate JAXB Classes . . . . .	572
Compiling JAXB Class Files . . . . .	573
About XML Schema Properties . . . . .	573
About xsd:schema Properties . . . . .	574
Element and Element Reference Properties in XML Schemas . . . . .	576
Attribute and Attribute Reference Properties in XML Schemas . . . . .	578
Group Properties in XML Schemas . . . . .	580
Model Group Properties in XML Schemas . . . . .	580
Complex and simpleType Properties in XML Schemas . . . . .	582
Restriction and Extension Type Properties in XML Schemas . . . . .	583
Content Type Properties in XML Schemas . . . . .	583
Aggregator Type Properties in XML Schemas . . . . .	584
Facet Type Properties in XML Schemas . . . . .	585
Notation Type Properties in XML Schemas . . . . .	586
Include Type Properties in XML Schemas . . . . .	586
Import Type Properties in XML Schemas . . . . .	587
Redefine Type Properties in XML Schemas . . . . .	587
Identity Constraint Type Properties in XML Schemas . . . . .	587
Constraint Element Type Properties in XML Schemas . . . . .	588
Documentation Type Properties in XML Schemas . . . . .	588

<b>Chapter 8: Defining Document Type Definitions</b> .....	589
What Is a DTD? .....	589
Creating DTDs .....	590
About Editing DTDs .....	591
Restrictions .....	591
About Modifiers in Element Definitions in DTDs .....	591
Description of Element Modifiers in DTDs .....	592
Simple Example of Aggregating Modifiers in DTDs .....	593
More Complex Example of Aggregating Modifiers in DTDs .....	593
Aggregating Modifiers to Allow Any Order and Any Number in DTDs .....	594
Defining Elements in DTDs .....	595
Defining Elements in the DTD Tree Tab .....	596
Specifying That an Element Can Have an Attribute in DTDs .....	596
Specifying That an Element is Required in DTDs .....	597
Specifying That an Element is Optional in DTDs .....	598
Specifying That Multiple Instances of An Element Are Allowed in DTDs .....	599
Specifying That An Element Can Contain One of a Group of Elements in DTDs .....	601
Specifying That an Element Can Contain One or More Elements in DTDs .....	602
Specifying That an Element Can Contain Data in DTDs .....	603
Moving, Renaming, and Deleting Elements in DTDs .....	603
Defining General Entities and Parameter Entities in DTDs .....	604
Steps for Defining Entities in DTDs .....	605
General Entity Example in a DTD .....	606
Parameter Entity Example in a DTD .....	606
Inserting White Space in DTDs .....	606
Adding Comments to DTDs .....	607
About Node Properties in DTDs .....	607
Description of Element Properties in DTDs .....	608
Description of Attribute Properties in DTDs .....	608
Description of Entity and Parameter Entity Properties in DTDs .....	610
Associating an XML Document with an External DTD .....	611
Moving an Internal DTD to an External File .....	611
<b>Chapter 9: Writing XPath Expressions</b> .....	613
About the XPath Processor .....	614
Where You Can Specify Queries .....	614
About XPath .....	614
Benefits of XPath .....	615

---

Internationalization . . . . .	616
Restrictions on Queries . . . . .	616
Sample Data for Examples and Testing . . . . .	616
About XML Document Structure . . . . .	617
A Sample XML Document . . . . .	618
Tree Representation of a Sample XML Document . . . . .	618
Steps for Trying the Sample Queries . . . . .	620
Getting Started with Queries . . . . .	621
Obtaining All Marked-Up Text . . . . .	621
Obtaining a Portion of an XML Document . . . . .	622
Obtaining All Elements of a Particular Name . . . . .	623
Obtaining All Elements of a Particular Name from a Particular Branch . . . . .	624
Different Results from Similar Queries . . . . .	625
Queries That Return More Than You Want . . . . .	625
Specifying Attributes in Queries . . . . .	626
Restrictions . . . . .	627
Attributes and Wildcards . . . . .	627
Filtering Results of Queries . . . . .	627
Quotation Marks in Filters . . . . .	628
More Filter Examples . . . . .	628
How the XPath Processor Evaluates a Filter . . . . .	629
Multiple Filters . . . . .	629
Filters and Attributes . . . . .	630
Wildcards in Queries . . . . .	630
Restrictions . . . . .	630
Attributes . . . . .	631
Calling Functions in Queries . . . . .	631
Case Sensitivity and Blank Spaces in Queries . . . . .	632
Precedence of Query Operators . . . . .	633
Specifying the Nodes to Evaluate . . . . .	634
Understanding XPath Processor Terms . . . . .	635
Axis . . . . .	635
Context Node . . . . .	635
Context Node Set . . . . .	635
Current Node . . . . .	635
Document Element . . . . .	635
Filter . . . . .	636
Location Path Expression . . . . .	636

## Contents

---

Location Step . . . . .	636
Node Test . . . . .	636
Root Node . . . . .	636
Starting at the Context Node. . . . .	637
About Root Nodes and Document Elements . . . . .	637
Starting at the Root Node . . . . .	637
Descending Along Branches. . . . .	638
Explicitly Specifying the Current Context . . . . .	639
Specifying Children or Descendants of Parent Nodes . . . . .	640
Examples of XPath Expression Results . . . . .	640
Syntax for Specifying an Axis in a Query . . . . .	641
Supported Axes. . . . .	642
About the child Axis . . . . .	642
About the descendant Axis. . . . .	643
About the parent Axis . . . . .	643
About the ancestor Axis . . . . .	643
About the following-sibling Axis. . . . .	644
About the preceding-sibling Axis . . . . .	644
About the following Axis. . . . .	644
About the preceding Axis. . . . .	645
About the attribute Axis . . . . .	645
About the namespace Axis. . . . .	646
About the self Axis. . . . .	646
About the descendant-or-self Axis. . . . .	646
About the ancestor-or-self Axis . . . . .	647
Axes That Represent the Whole XML Document . . . . .	647
Handling Strings and Text . . . . .	648
Searching for Strings . . . . .	648
Finding Identical Strings . . . . .	648
Case Sensitivity . . . . .	649
Finding Strings That Contain Strings You Specify . . . . .	649
Finding Substrings That Appear Before Strings You Specify . . . . .	649
Finding Substrings That Appear After Strings You Specify . . . . .	650
Finding Substrings by Position . . . . .	650
Manipulating Strings . . . . .	651
Concatenating Strings . . . . .	651
Determining the Number of Characters in a String . . . . .	651
Normalizing Strings . . . . .	652
Replacing Characters in Strings with Characters You Specify . . . . .	652



---

Converting Objects to Strings . . . . .	653
Finding Strings That Start with a Particular String . . . . .	654
Obtaining the Text Contained in a Node . . . . .	654
Specifying Boolean Expressions and Functions . . . . .	655
Using Boolean Expressions . . . . .	655
Case Sensitivity . . . . .	655
Examples . . . . .	655
Calling Boolean Functions . . . . .	656
Converting an Object to Boolean . . . . .	656
Obtaining Boolean Values . . . . .	657
Determining the Context Node Language . . . . .	657
Specifying Number Operations and Functions . . . . .	658
Performing Arithmetic Operations . . . . .	658
Calling Number Functions . . . . .	659
Converting an Object to a Number . . . . .	659
Obtaining the Sum of the Values in a Node Set . . . . .	660
Obtaining the Largest, Smallest, or Closest Number . . . . .	660
Comparing Values . . . . .	661
About Comparison Operators . . . . .	662
How the XPath Processor Evaluates Comparisons . . . . .	662
Comparing Node Sets . . . . .	663
Two Node Sets . . . . .	663
A Node Set and a Number . . . . .	663
A Node Set and a String . . . . .	664
A Node Set and a Boolean Value . . . . .	664
Comparing Single Values With = and != . . . . .	664
Comparing Single Values With <=, <, >, and >= . . . . .	665
Priority of Object Types in Comparisons . . . . .	665
Examples of Comparisons . . . . .	666
Operating on Boolean Values . . . . .	666
Finding a Particular Node . . . . .	666
About Node Positions . . . . .	667
Determining the Position Number of a Node . . . . .	667
Positions in Relation to Parent Nodes . . . . .	668
Finding Nodes Relative to the Last Node in a Set . . . . .	669
Finding Multiple Nodes . . . . .	669
Examples of Specifying Positions . . . . .	670
Finding the First Node That Meets a Condition . . . . .	670
Finding an Element with a Particular ID . . . . .	671

## Contents

---

The id() Function's Argument . . . . .	671
Unique IDs . . . . .	671
Obtaining Particular Types of Nodes By Using Node Tests . . . . .	672
About the Document Object. . . . .	673
Getting Nodes of a Particular Type . . . . .	673
Obtaining a Union . . . . .	674
Obtaining Information About a Node or a Node Set . . . . .	675
Obtaining the Name of a Node . . . . .	675
Wildcards . . . . .	675
Obtaining Namespace Information. . . . .	675
Obtaining the Namespace URI. . . . .	676
Obtaining the Local Name . . . . .	676
Obtaining the Expanded Name . . . . .	676
Specifying Wildcards with Namespaces . . . . .	677
Examples of Namespaces in Queries . . . . .	677
Obtaining the URI for an Unparsed Entity . . . . .	678
Determining the Number of Nodes in a Collection . . . . .	678
Determining the Context Size. . . . .	678
Using XPath Expressions in Stylesheets. . . . .	679
Using Variables . . . . .	679
Obtaining System Properties . . . . .	679
Determining If Functions Are Available . . . . .	680
Obtaining the Current Node for the Current XSLT Template . . . . .	680
Finding an Element with a Particular Key . . . . .	681
Generating Temporary IDs for Nodes . . . . .	683
Format . . . . .	683
Accessing Other Documents During Query Execution . . . . .	683
Format of the document() Function . . . . .	684
When the First Argument is a Node Set. . . . .	684
Specification of Second Argument. . . . .	684
Example of Calling the document() Function . . . . .	685
XPath Quick Reference. . . . .	685
XPath Functions Quick Reference . . . . .	686
XPath Syntax Quick Reference . . . . .	689
Axes . . . . .	690
Node Tests . . . . .	690
Filters . . . . .	690
Location Steps . . . . .	691
XPath Expression. . . . .	691

---

XPath Abbreviations Quick Reference .....	691
<b>Chapter 10: Working with XQuery in Stylus Studio</b> .....	<b>695</b>
Getting Started with XQuery in Stylus Studio .....	695
What is XQuery? .....	696
Example .....	696
Sources for Additional XQuery Information .....	696
What is an XQuery? .....	696
The Stylus Studio XQuery Editor .....	696
XQuery Source Tab .....	697
Mapper Tab .....	699
XQuery Source and Mapper Tab Interaction .....	700
Building an XQuery Using the Mapper .....	700
Process Overview .....	701
Working with Existing XQueries .....	701
Source Documents .....	701
Choosing Source Documents .....	702
Source Documents and XML Instances .....	702
Source document icons .....	704
How to Change a Source Document Association .....	704
How to Add a Source Document .....	704
How to Remove a Source Document .....	706
How Source Documents are Displayed .....	706
Document structure symbols .....	707
Getting source document details .....	707
Specifying a Target Structure .....	707
Using an Existing Document .....	708
Building a Target Structure .....	708
Modifying the Target Structure .....	709
Adding a Node .....	709
Removing a Node .....	710
Setting a Text Value .....	710
Mapping Source and Target Document Nodes .....	711
Preserving Mapper Layout .....	711
Left and Right Mouse Buttons Explained .....	711
How to Map Nodes .....	712
Link Lines Explained .....	713
Removing Source-Target Map .....	716

## Contents

---

FLWOR Blocks . . . . .	716
Parts of a FLWOR Block . . . . .	717
Creating a FLWOR Block . . . . .	718
Function Blocks . . . . .	719
Standard Function Block Types . . . . .	719
Creating a Function Block . . . . .	719
Parts of a Function Block . . . . .	720
User-Defined Functions . . . . .	721
concat Function Blocks . . . . .	722
IF Blocks . . . . .	722
Condition Blocks . . . . .	723
Debugging XQuery Documents . . . . .	724
Using Breakpoints . . . . .	725
Inserting Breakpoints . . . . .	725
Removing Breakpoints . . . . .	725
Start Debugging . . . . .	725
Viewing Processing Information . . . . .	726
Watching Particular Variables . . . . .	726
Evaluating XPath Expressions in the Current Processor Context . . . . .	726
Obtaining Information About Local Variables . . . . .	727
Displaying a List of Process Suspension Points . . . . .	727
Displaying XQuery Expressions for Particular Output . . . . .	727
Using Bookmarks . . . . .	728
Inserting . . . . .	728
Removing . . . . .	728
Moving Focus . . . . .	728
Profiling XQuery Documents . . . . .	728
About Performance Metrics . . . . .	729
Enabling the Profiler . . . . .	730
Displaying the XQuery Profiler Report . . . . .	731
Creating an XQuery Scenario . . . . .	731
Overview of Scenario Features . . . . .	732
Specifying XML Input . . . . .	732
Selecting an XQuery Processor . . . . .	733
Setting Values for External Variables . . . . .	734
Performance Metrics Reporting . . . . .	735
Validating XQuery Results . . . . .	735
How to Create a Scenario . . . . .	737
How to Run a Scenario . . . . .	738

---

How to Clone a Scenario . . . . .	738
Generating Java Code for XQuery . . . . .	739
Scenario Settings . . . . .	739
Choosing Scenarios . . . . .	740
Java Code Generation Settings . . . . .	741
How to Generate Java Code for XQuery . . . . .	742
Compiling Generated Code . . . . .	743
How to Modify the Stylus Studio Classpath . . . . .	743
How to Compile and Run Java Code in Stylus Studio . . . . .	744
<b>Chapter 11: Composing Web Service Calls . . . . .</b>	<b>745</b>
Overview . . . . .	746
How to Compose a Web Service Call . . . . .	746
Obtaining WSDL URLs . . . . .	748
Browsing UDDI Registries . . . . .	749
How to Browse UDDI Registries . . . . .	751
Modifying a SOAP Request . . . . .	752
Understanding Parameters . . . . .	752
Displaying a WSDL Document . . . . .	753
How to Modify a SOAP Request . . . . .	753
Testing a Web Service . . . . .	754
What Happens When You Test a Web Service . . . . .	754
Other Options for Testing a Web Service . . . . .	754
How to Test a Web Service . . . . .	755
Saving a Web Service Call . . . . .	755
Using Web Service Calls as XML . . . . .	756
Where Web Service Calls are Saved . . . . .	758
How to Save a Web Service Call . . . . .	758
Creating a Web Service Call Scenario . . . . .	758
Overview of Scenario Features . . . . .	759
Scenario Names . . . . .	760
Transport Protocol and Client Settings . . . . .	761
Other Transport Settings . . . . .	761
How to Create a Scenario . . . . .	762
How to Run a Scenario . . . . .	763
How to Clone a Scenario . . . . .	764

**Chapter 12: Accessing Relational Data as XML** ..... 765

- Overview of DB-to-XML Data Sources ..... 766
  - System Requirements ..... 766
  - Supported Databases ..... 766
  - DB-to-XML Data Source Scenarios ..... 767
  - The DB-to-XML Data Source Editor ..... 768
    - SQL/XML Editor Pane ..... 768
    - Database Schema Pane ..... 769
    - Working with Relational Data ..... 770
- Creating a DB-to-XML Data Source ..... 770
  - Saving a DB-to-XML Data Source ..... 771
  - Opening a DB-to-XML Data Source ..... 771
- Specifying Connection Settings ..... 773
  - Choosing a Database ..... 773
    - Using SequeLink ..... 774
  - Using the Server URL Field ..... 774
  - Username and Password ..... 775
  - Creating a Default Database Connection ..... 775
- Working with Scenarios ..... 776
  - Opening the Scenario Properties Dialog Box ..... 776
  - Creating a Scenario ..... 777
    - Naming Scenarios ..... 777
    - How to Create a Scenario ..... 777
    - Alternative ..... 778
  - How to Modify a Scenario ..... 779
  - How to Clone a Scenario ..... 779
  - How to Rename a Scenario ..... 780
  - How to Delete a Scenario ..... 780
- Composing SQL/XML in Stylus Studio ..... 780
  - Manually Entering SQL/XML ..... 781
  - Using Drag-and-Drop ..... 781
    - Example ..... 782
  - How Relational Data is Translated to XML ..... 782
- Working with Relational Data as XML ..... 784
  - Understanding SELECT and UPDATE ..... 784
    - When Statements are Executed ..... 785
  - Example ..... 785
    - The SELECT Statement ..... 785
    - The INSERT Statement ..... 786

---

Saving the File . . . . .	786
Opening the .rdbxml as XML . . . . .	787
Updating the Data in the Database . . . . .	788
Using the URL Builder . . . . .	789
URL Builder Compared to DB-to-XML Data Sources . . . . .	789
When to Use . . . . .	790
Connection Settings . . . . .	790
XMLForest . . . . .	790
Example . . . . .	791
Sequence Elements . . . . .	791
How to Open a Table or View as an XML Document . . . . .	792
<b>Chapter 13: Extending Stylus Studio . . . . .</b>	<b>795</b>
Custom XML Validation Engines . . . . .	795
Registering a Custom Validation Engine . . . . .	796
Configuring a Custom Validation Engine . . . . .	797
The Custom Validation Engines Page . . . . .	797
How to Configure a Custom Validation Engine . . . . .	800
Custom Document Wizards . . . . .	801
Registering a Custom Document Wizard . . . . .	802
Configuring a Custom Document Wizard . . . . .	802
The Custom Document Wizards Page . . . . .	803
Defining Arguments . . . . .	807
How to Configure a Custom Document Wizard . . . . .	811
Stylus Studio File System Java API . . . . .	811
Overview . . . . .	812
Javadoc for the Stylus Studio File System Java API . . . . .	813
Custom File Systems . . . . .	813
Creating a Custom File System . . . . .	813
File System Interfaces . . . . .	814
Examples . . . . .	814
Registering a Custom File System . . . . .	815
The Custom File Systems Page . . . . .	815
How to Display . . . . .	816
Fields . . . . .	816
How to Register a Custom File System . . . . .	817

## Contents

---

Using Stylus Studio with Berkeley DB XML.....	817
Overview.....	818
Berkeley DB XML Support.....	818
Usage Summary.....	819
Prerequisites.....	819
Saving Files to Berkeley DB XML.....	820
Configuring the BerkeleyDBXML.xml File.....	820
The Berkeley Database Environment.....	820
The BerkeleyDBXML.xml Configuration File.....	821
How to Configure BerkeleyDBXML.xml.....	822
Open the Output Window.....	822
Naming Files.....	823
Example – Berkeley DB XML V 1.2.1.....	823
Opening a File Stored on Berkeley DB XML.....	824
Saving a File Back to Berkeley DB XML.....	824
Saving and Containers.....	824
Validating Documents.....	825
Creating Containers with Stylus Studio.....	826
Node and Wholedoc Containers.....	826
Usage Tips.....	827
Before Starting Stylus Studio.....	827
Database Environment Recovery.....	827
<b>Index.....</b>	<b>829</b>



---

## 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®.
- [Technical Support](#) provides information on contacting Technical Support.

## About This Manual

This manual describes how to use Stylus Studio 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®,”](#) 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 the Stylus Studio Convert to XML module to create converters that allow you to open any flat file (.txt, binary, or EDI, for example) as an XML document.

- [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,”](#) discusses 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 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, “Accessing Relational Data as XML,”](#) describes how to use Stylus Studio to connect to a relational database, and how to query and update relational data.
- [Chapter 13, “Extending Stylus Studio,”](#) provides a description of advanced Stylus Studio features, including information about using the Stylus Studio custom document wizard.

---

## 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:



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

See [Stylus Studio Editions](#) on page 2 for more information about the features that are available in a given edition.

## 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**

<i>Document</i>	<i>Description</i>
<i>Stylus Studio® 6 User Guide</i>	Describes how to use Stylus Studio to develop XML applications using XML, SQL/XML, XQuery, Web Services, and XSLT.
<i>Release Notes</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 <b>Help &gt; Documentation</b> from the menu bar. You can also view the help independently of the application, by opening <code>ide.chm</code> in the \doc directory where you installed Stylus Studio.

## Technical Support

Submit questions and report problems using the [Stylus Studio Developer Network \(SSDN\)](#). Once you log in, navigate to the Stylus Studio Technical Forum, and enter your issue there.

When submitting an issue, consider providing the following information:

- The release version number and serial number of your copy of Stylus Studio. This information is listed in the **About Stylus Studio** dialog box (click **Help > About Stylus Studio** on the menu bar).
- Your name and, if applicable, your company name.
- E-mail address, telephone, and fax numbers for contacting you.
- The platform on which you are running Stylus Studio, as well as any other environment information you think might be relevant, such as the Java Virtual Machine (JVM) you are using.



# Chapter 1 Getting Started with Stylus Studio®

Stylus Studio® 6 (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.

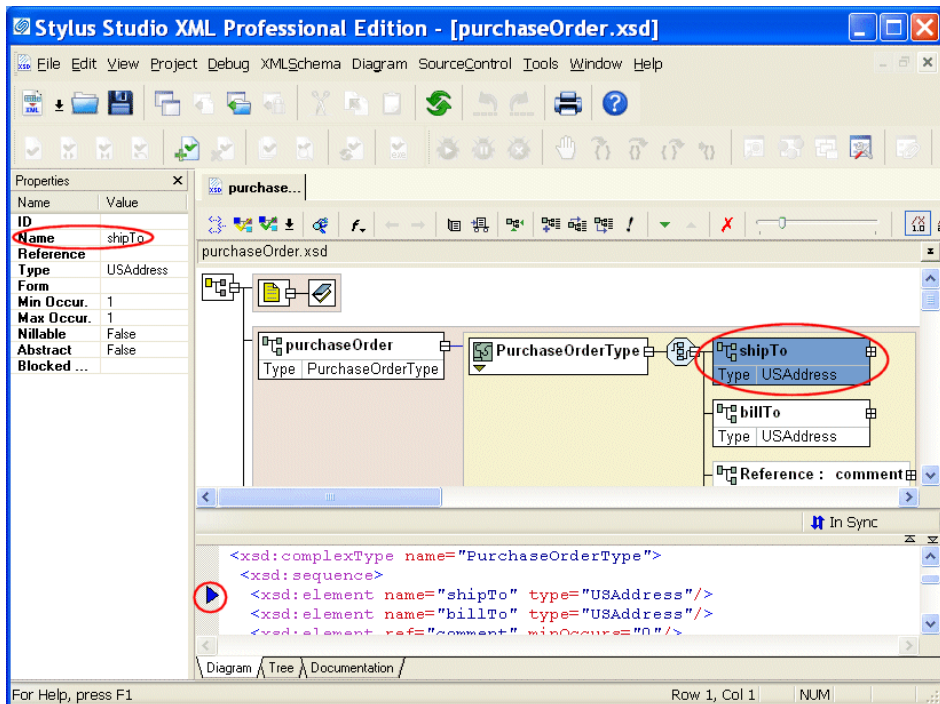


Figure 1. Stylus Studio's XML Schema Diagram Editor

Stylus Studio includes modules for:

- XML
- XQuery
- XSLT
- DTD
- XML Schema
- SQL/XML
- Web services
- Java
- Converting non-XML files to XML using built-in and user-defined adapters

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

## 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 Edition

Stylus Studio XML Enterprise Edition 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 Edition 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
- Web Service Call Composer to help you build and test calls to hundreds of industry-standard Web services
- Integration with Mark Logic
- Support for OASIS catalogs, including dozens of catalogs bundled with Stylus Studio



---

## Stylus Studio XML Professional Edition

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

- Convert to XML, a module that allows you to easily convert non-XML files to XML
- XML Differencing for comparing multiple XML documents and folders
- DB-to-XML Data Sources, a module that allows you to access relational data and render it as XML
- Integration with Sleepycat Software's Berkeley DB XML
- A WYSIWYG HTML designer for XSLT
- Custom file systems

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

## 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 Edition, allowing you to do real work with XML, XML Schema, XSLT, DTD, and other important XML technologies.

## Edition Alerts

The *Stylus Studio® 6 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 Edition and Stylus Studio XML Professional Edition.

## More Information

For a complete description of Stylus Studio XML Enterprise, XML Professional, and Home editions, see the Stylus Studio Web site:

[http://www.StylusStudio.com/xml\\_feature\\_comparison.html](http://www.StylusStudio.com/xml_feature_comparison.html)

### In This Chapter

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:

- [“Starting Stylus Studio”](#) on page 5
- [“Updating an XML Document—Getting Started”](#) on page 6
- [“Working with Stylesheets – Getting Started”](#) on page 26
- [“Stylesheets That Generate HTML – Getting Started”](#) on page 47
- [“Using the XSLT Mapper – Getting Started”](#) on page 64
- [“Debugging Stylesheets – Getting Started”](#) on page 71
- [“Defining a DTD – Getting Started”](#) on page 80
- [“Defining an XML Schema Using the Diagram Tab – Getting Started”](#) on page 85
- [“Opening Files in Stylus Studio”](#) on page 110
- [“Working with Projects”](#) on page 118
- [“Customizing Tool Bars”](#) on page 136
- [“Specifying Stylus Studio Options”](#) on page 138
- [“Defining Keyboard Shortcuts”](#) on page 145
- [“Using Stylus Studio from the Command Line”](#) on page 147
- [“Managing Stylus Studio Performance”](#) on page 151

## 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 Edition, you would

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

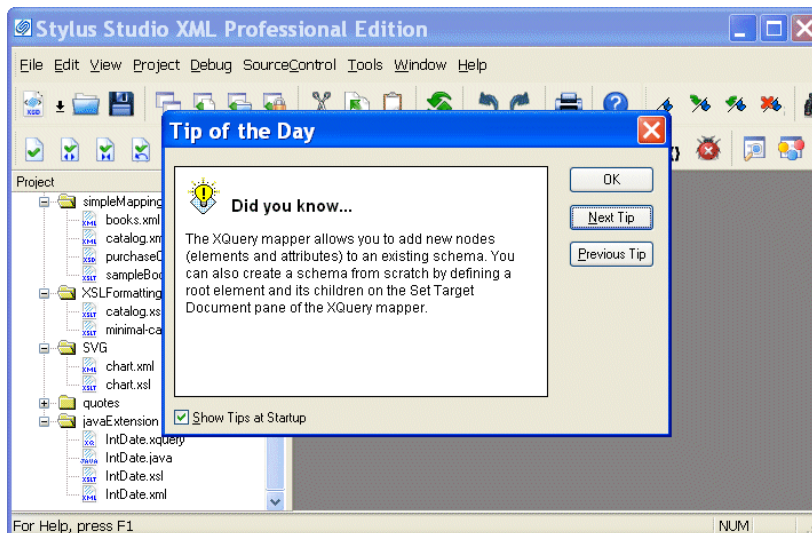
The path shown here assumes that you accepted the defaults when you installed Stylus Studio. If you did not, you must alter your selection path accordingly.

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 Edition 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 Professional Edition > Documentation**.

## 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 6
- “Updating the Text of a Sample Document” on page 8
- “Updating the Schema of a Sample Document” on page 14
- “Updating the Tree Representation of a Sample Document” on page 20
- “Updating a Sample Document Using the Grid Tab” on page 23

### Opening a Sample XML Document

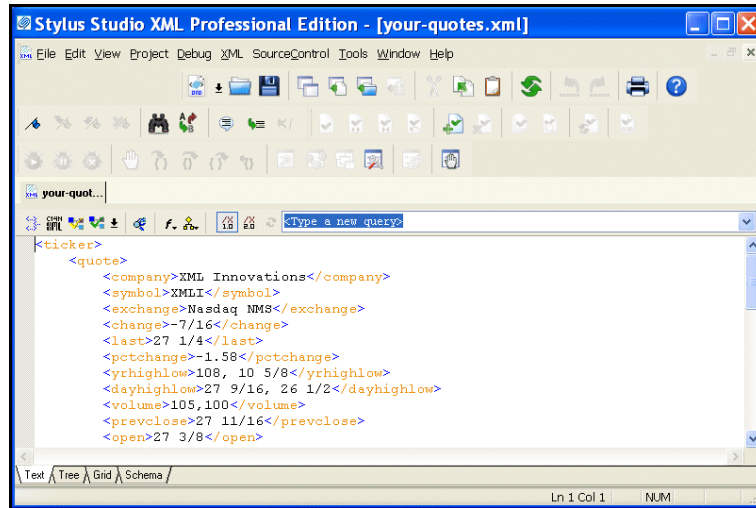
◆ **To open the `your-quotes.xml` sample XML document in Stylus Studio:**

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 110 to learn more about the **File Explorer** window.

See [“Working with Projects”](#) on page 118 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 [Updating XML Documents Using the Text Editor](#) on page 166.

This section provides instructions for

- [“Displaying Line Numbers”](#) on page 8
- [“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 a Query”](#) 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**.

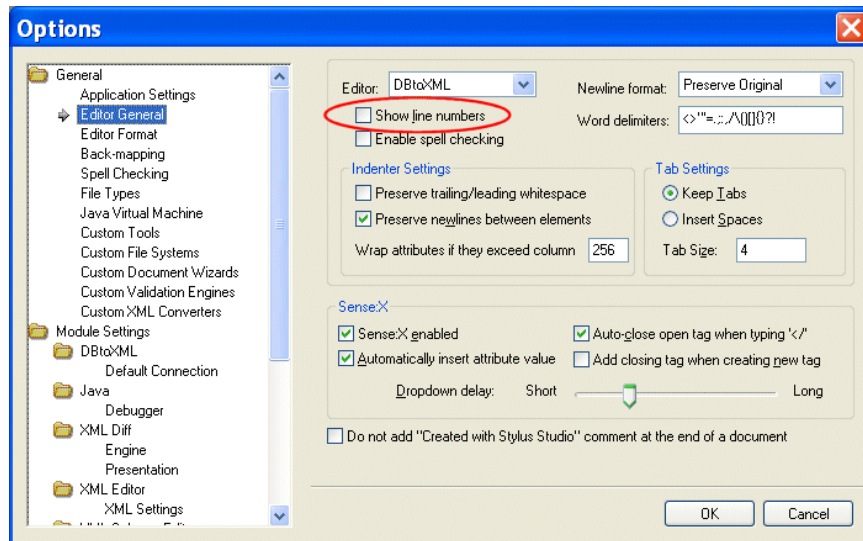


Figure 5. Sense:X and Other Editor Features are in the Options Dialog Box

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.

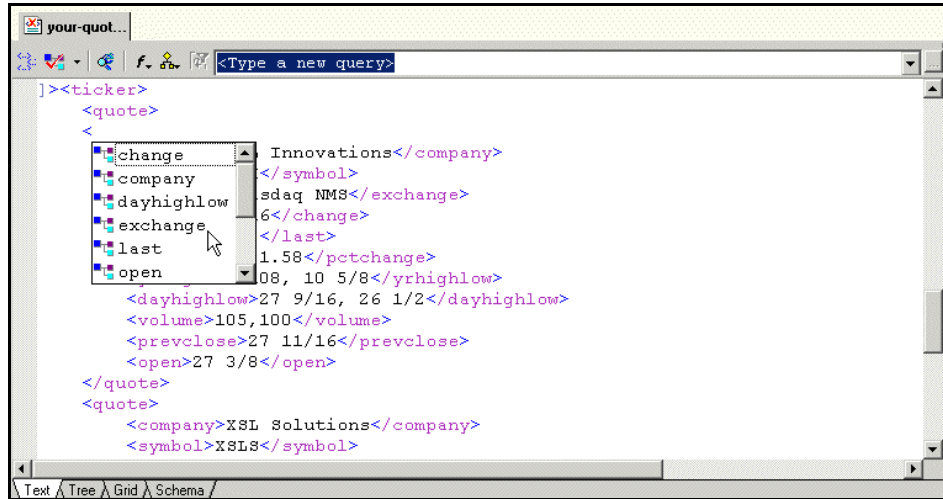



Figure 6. Example of Stylus Studio Sense:X

## 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

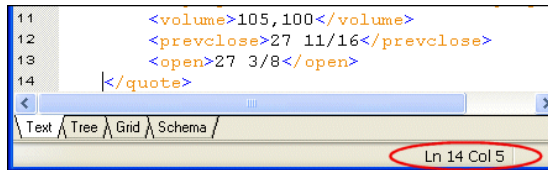
◆ **To insert indents in `your-quotes.xml`:**

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**

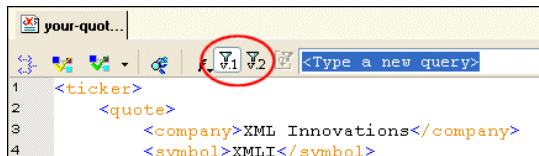
3. In line 2, click after the </company> tag and type </>. By default, Stylus Studio displays **quote>** because it is the only element that is appropriate to close.
4. In the XML Editor tool bar, click **Indent XML Tags** .

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

Stylus Studio provides support for both XPath 1.0 and 2.0. You control the version you want Stylus Studio to use to process your XPath using the **v.1** and **v.2** buttons in the XML Editor tool bar. The default XPath used is version 1.0.



**Figure 8. Specify XPath Version with a Single Click**

◆ **To query** your-quotes.xml:

1. Click in the **<Type a new query>** field at the top of the XML Editor.  
*Alternative:* Press F6. This moves the focus immediately to the query definition field.

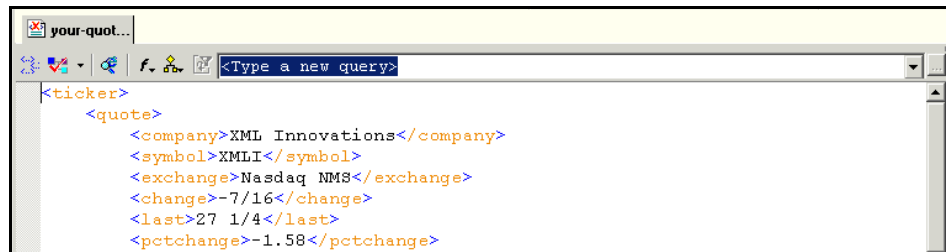


Figure 9. The XML Editor's Query Definition Field

2. Type `/ticker/quote` and press Enter.  
Stylus Studio runs the `/ticker/quote` query on `your-quotes.xml`, and displays the results in the **Query Output** window.
3. In the **Query Output** window, expand the second **quote** element to view its contents.

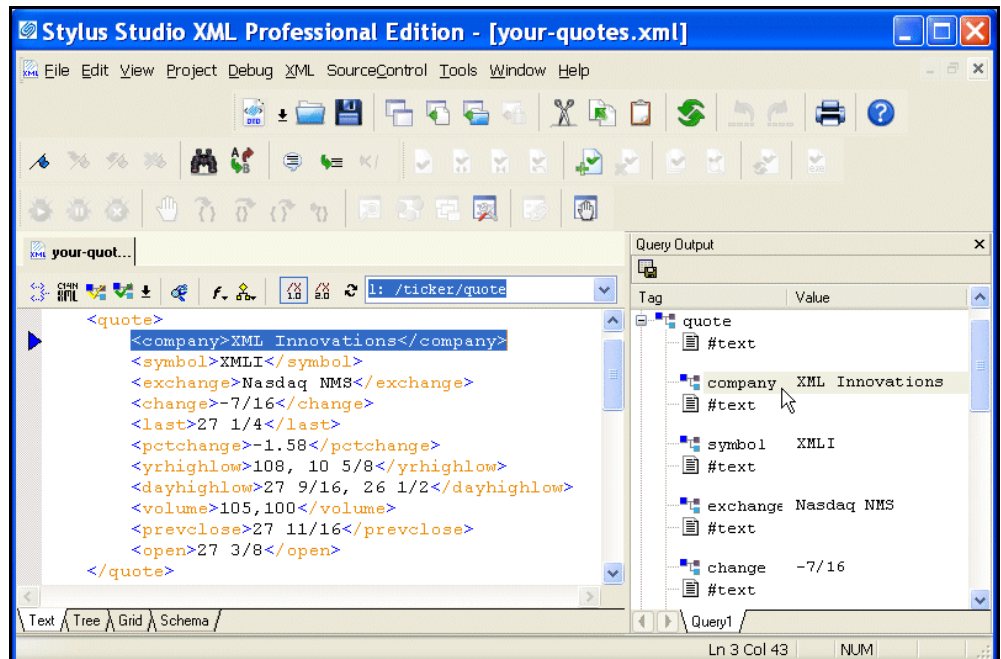



Figure 10. Query Result Displayed in Query Output Window

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.
5. In the **Text** view, click the down arrow to the right of the query field.
6. Click **<type a new query>**.
7. Type `//company` and press Enter.  
Stylus Studio runs the new query and displays the results on a new tab (labelled **Query2**) in the **Query Output** window. Stylus Studio adds a new tab for each query you define.
8. Click the **Query1** tab to view the results for the first query.
9. Click **Save Result**  at the top of the **Query Output** window.  
Stylus Studio displays the **Save As** dialog box.
10. In the **URL:** field, type `query1.xml` and press Enter.  
This saves the results from the `/ticker/quote` query in `query1.xml`.
11. Close the **Query Output** window by clicking the **x** in that window's upper right corner.

### Deleting a Query

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.

**Note** See [“Working with Projects”](#) on page 118 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:

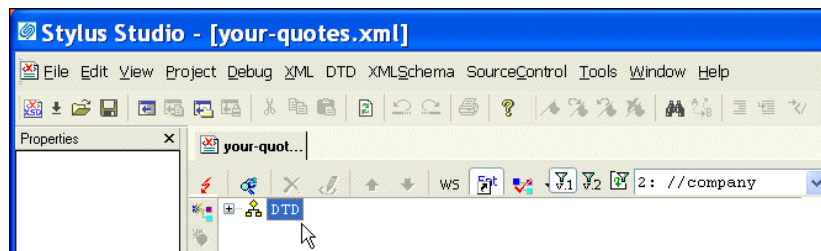
- “Creating a Sample Schema” on page 15
- “Defining a Sample Element” on page 17
- “Adding an Element Reference to a Sample Schema” on page 18
- “Defining an Entity in a Sample Schema” on page 19
- “Exploring Other Features in a Sample Schema” on page 19

For more information, see “Defining a DTD – Getting Started” on page 80.

### 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 6 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.

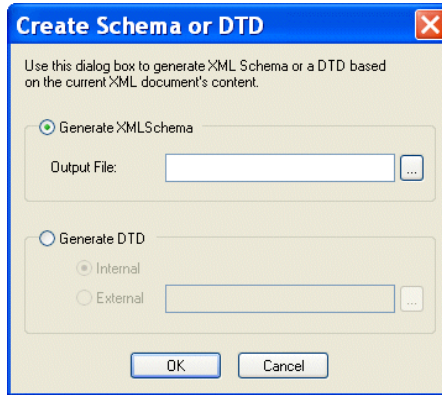


**Figure 11. Default Schema Tab for Document With no Schema**

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

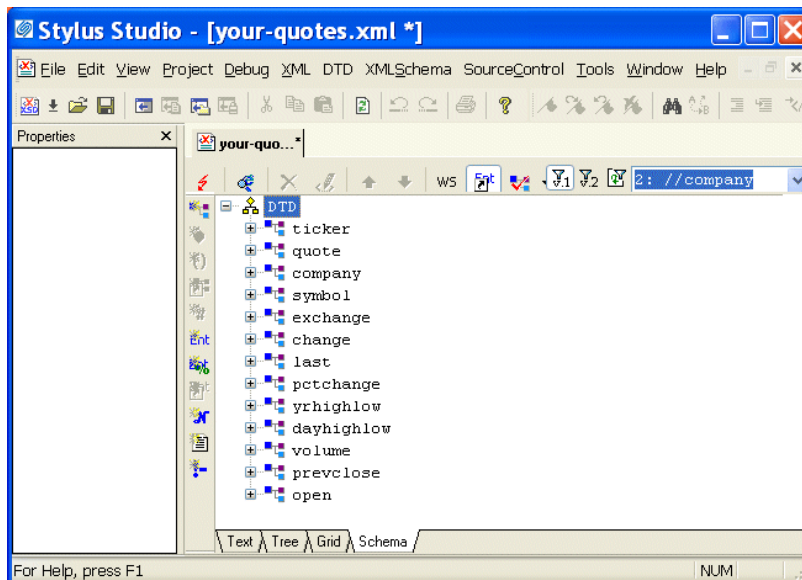
of the document. You can also use this dialog box to generate an external XML Schema or DTD.



**Figure 12. 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.



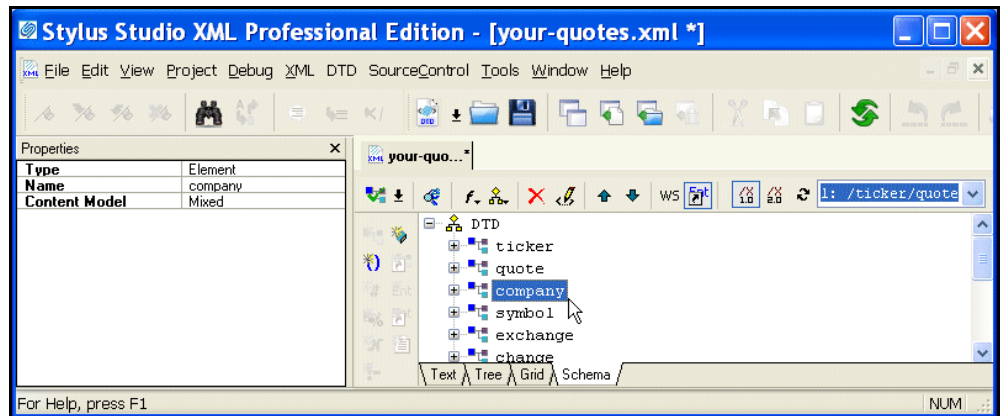
**Figure 13. Result of Generating a Schema Based on XML Content**

## 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 14. 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.

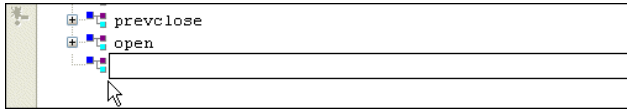
**Tip**

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.

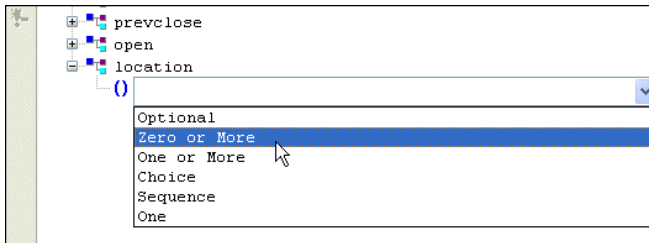
In the left tool bar, Stylus Studio activates only those buttons that are applicable to the DTD – you can add elements, entities, comments, and so on. But you cannot add an attribute definition, a reference to an element, or a **#PCDATA** node, for example.

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




**Figure 15. 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.
5. 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 16. Drop-Down List for Modifier Values**

6. Double-click **Zero or More** (or click once to select it and press Enter).
7. 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:**

1. In the **Schema** tab, expand the **quote** element.
2. Click its **Sequence** modifier.
3. 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.


4. In the drop-down list, double-click **Optional**.



5. In the left tool bar, click **New Reference to Element** . Stylus Studio displays an entry field after the new **Optional** modifier.
6. Type location and press Enter.
7. To move the **location** element to be earlier in the sequence, click its **Optional** modifier.
8. 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:**

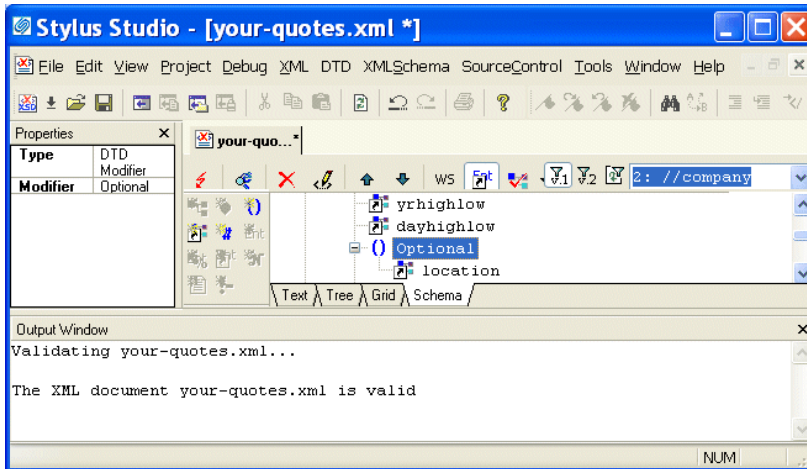
1. Click the **DTD** node.
2. In the left tool bar, click **New Entity** .  
At the end of the schema, Stylus Studio displays **Ent** and a entry field for the name of the new entity.
3. Type TCBC for the name of the entity and press Enter.  
In the **Properties** window, Stylus Studio displays the properties 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 17. 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 14. 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 21
- [“Changing an Element’s Data in a Sample Document Tree”](#) on page 21
- [“Adding Attributes and Other Node Types to a Sample Document Tree”](#) on page 22
- [“Adding an Entity Reference to a Sample Document Tree”](#) on page 23

## 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 6 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:**

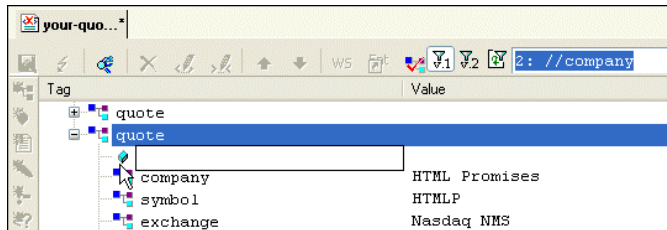
1. Expand the third **quote** element.
2. Click the **symbol** element.
3. In the XML Editor top tool bar, click **Change Value** .  
Stylus Studio activates the field to the right of the **symbol** element and selects the current value.
4. In the active field, type `XSOL` and press Enter.
5. To the 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 18. 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

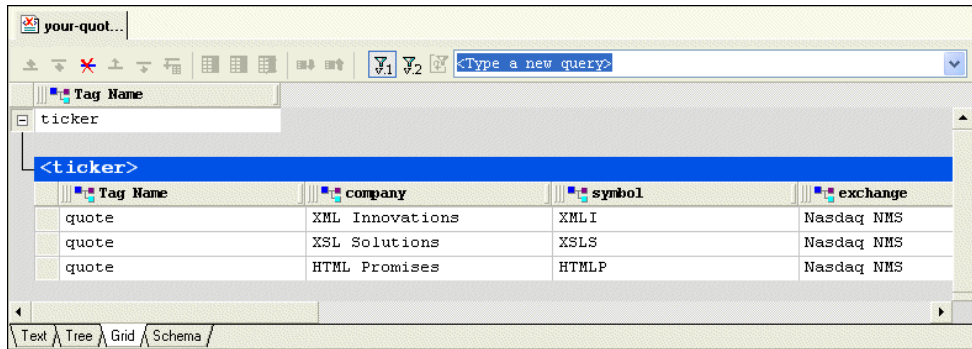
This section provides instructions for updating the `your-quotes.xml` document using the **Grid** tab of the XML Editor. The **Grid** tab is useful for displaying structured data. It is a convenient way to view a document that contains multiple instances of the same type of element, for example.



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

◆ **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 6 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 19. 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.

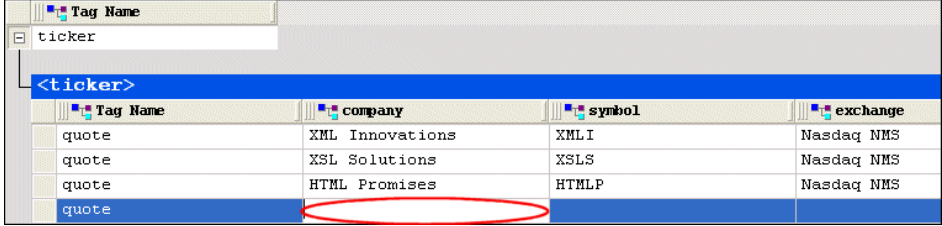
**Tip**

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.

- 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	XMLI	Nasdaq NMS
quote	XSL Solutions	XSLS	Nasdaq NMS
quote	HTML Promises	HTMLP	Nasdaq NMS
quote			

**Figure 20. Grid with a New Row**

- Type XML Designs and press Enter. Stylus Studio creates the value for the <company> subelement.
- Press Tab (or use the right arrow key) to move the cursor to the next cell in the row.
- Repeat [Step 5](#) and [Step 6](#) to create values for the <symbol> and <exchange> subelements.
- 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 generate HTML, see “[Stylesheets That Generate HTML – Getting Started](#)” on page 47. To focus on stylesheets that map XML to XML, see “[Using the XSLT Mapper – Getting Started](#)” on page 64.

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 26
- “[XSLT Stylesheet Editor Quick Tour](#)” on page 27
- “[XSLT Scenarios](#)” on page 33
- “[Making a Static Web Page Dynamic by Editing XSLT](#)” on page 38

Before you begin

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

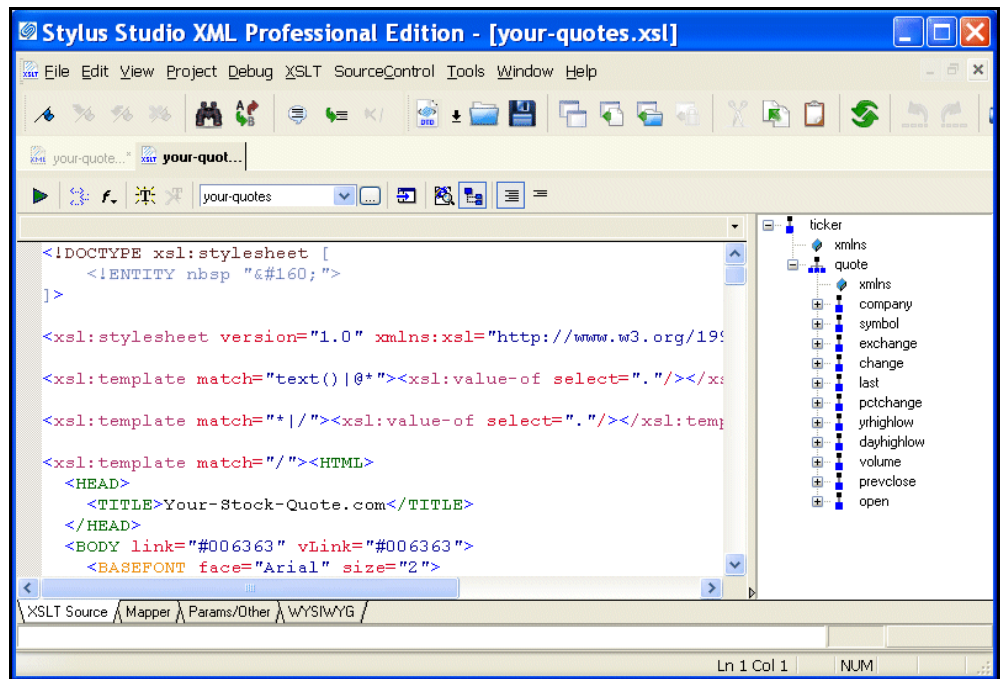
◆ **To open the `your-quotes.xml` sample XSLT stylesheet in Stylus Studio:**

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.xsl**. Stylus Studio displays the your-quotes.xsl document in the **XSLT Source** tab of the XSLT editor.



**Figure 21. 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 28
- “Exploring the XSLT Source Tab” on page 28
- “Exploring the Params/Other Tab” on page 31
- “Exploring the WYSIWYG 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 – you can see how changes you make to the XSLT using graphical editors, such as the **WYSIWYG** tab, affect the source.

**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 64.

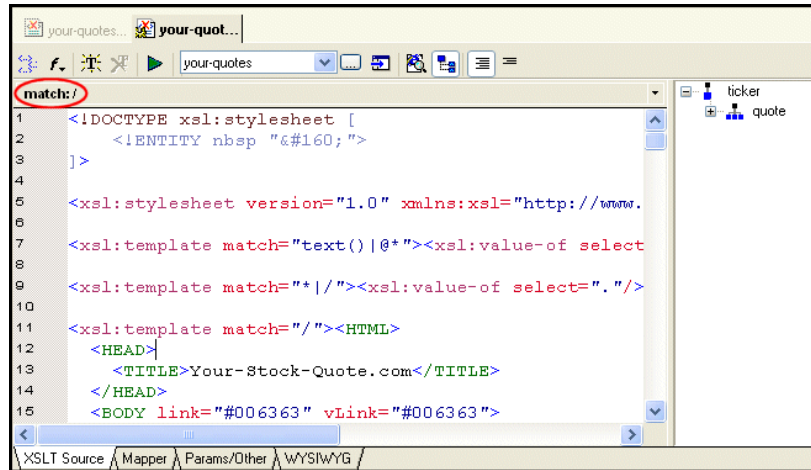
- **Params/Other.** You use the **Params/Other** tab to specify the encoding Stylus Studio uses to store the stylesheet, the stylesheet’s output method, and the encoding Stylus Studio uses for the document that results from applying this stylesheet. You can also use this tab to view default values for parameters used by your stylesheet.
- **WYSIWYG.** The **WYSIWYG** tab allows you to compose HTML by dragging and dropping XML document elements and attributes. The **WYSIWYG** tab tool bar lets you easily format text, create tables, build lists, and Stylus Studio automatically generates the XSLT instructions that output the content you define on the tab.

### 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 22. 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** .

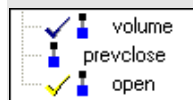
Stylus Studio inserts the following after the last template already specified in the stylesheet.

```
<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**

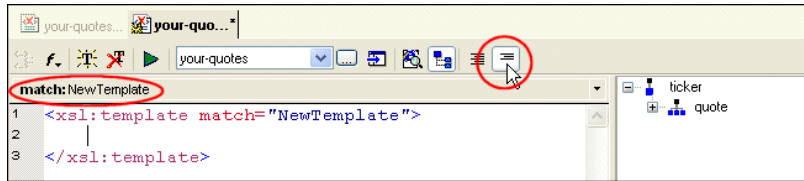
You can also create a new template by double-clicking a node on the schema tree. Templates that match nodes in the XSLT document are displayed with a check in the schema tree, as shown here.



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

4. In the XSLT Editor tool bar, click **Template Mode** , which is the right most button.

Stylus Studio displays only the new template.



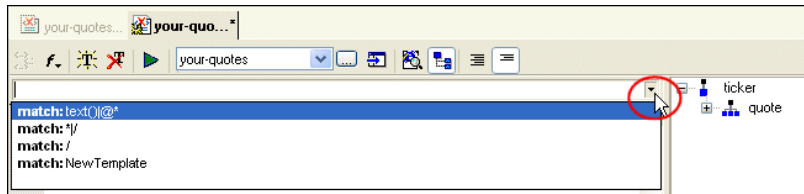
**Figure 23. Use Template Mode to Focus on a Single Template**

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.


5. 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 24. You Can Show Individual Templates in a Stylesheet**

6. 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 383.

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.

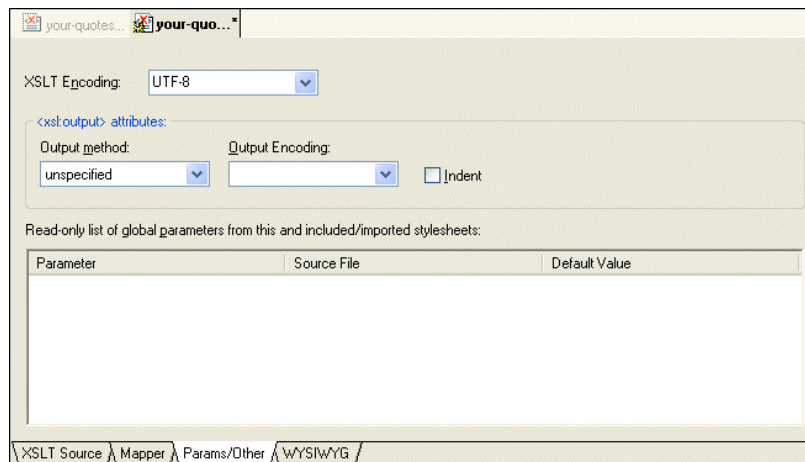
7. Click **Full Source Mode** .

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 25. 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.

## Exploring the WYSIWYG Tab



The XSLT Editor **WYSIWYG** tab is available only in Stylus Studio XML Professional Edition.

The **WYSIWYG** tab is a graphical XSLT editor that lets you compose XSLT by constructing the HTML document you want the XSLT to output.

◆ **To work with the WYSIWYG tab:**

1. Click the **WYSIWYG** tab.

Stylus Studio displays a message that indicates that the output of the stylesheet is unspecified and asks you if you want to make it HTML. When the output is HTML (or XML), you can edit the stylesheet in the WYSIWYG HTML editor.

2. Click **Yes** to make the stylesheet’s output HTML.

Stylus Studio displays the **WYSIWYG** tab.

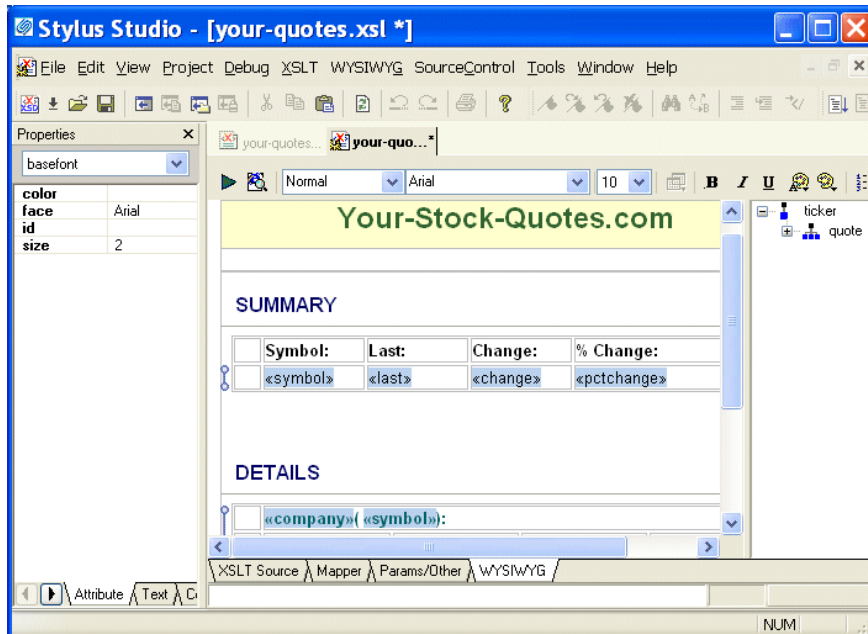
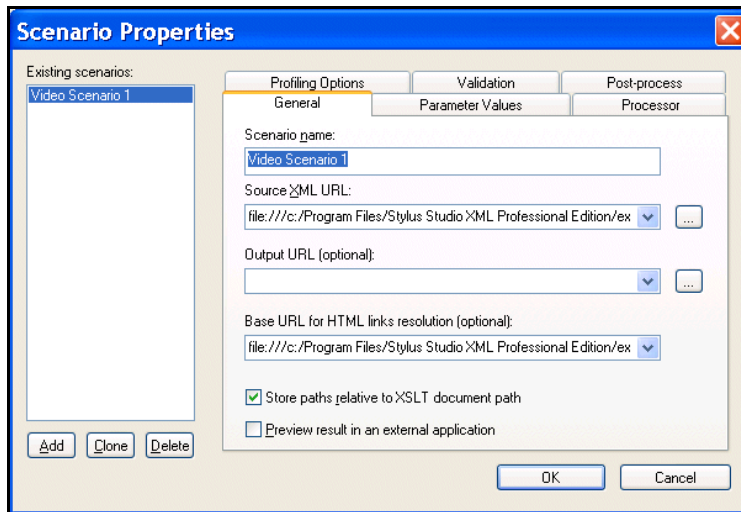


Figure 26. XSLT Editor WYSIWYG Tab

See “[Stylesheets That Generate HTML – Getting Started](#)” on page 47 to learn more about using the XSLT WYSIWYG editor.

## 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 27. 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, DB-to-XML Data Sources, and XQuery.

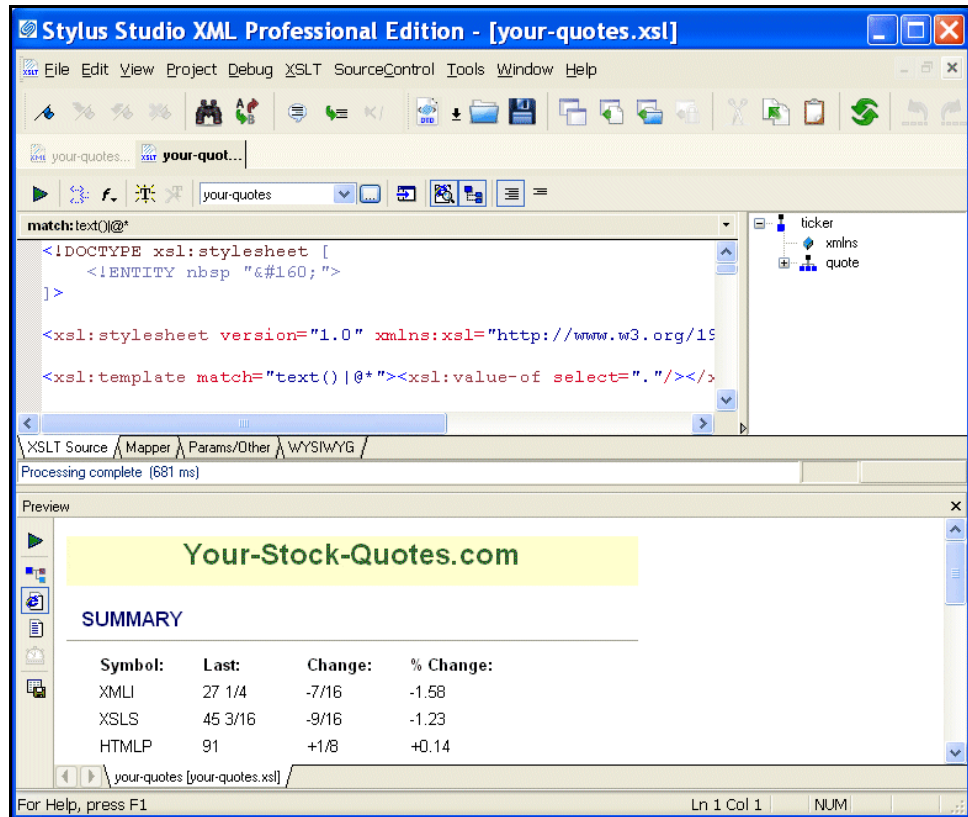
An XSLT scenario is defined by a single stylesheet-XML document pair. You can associate any number of scenarios with a stylesheet, though only one scenario can be in effect at the time the XSLT is processed. Similarly, you can associate any number of scenarios with an XML source document.

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

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 28. 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

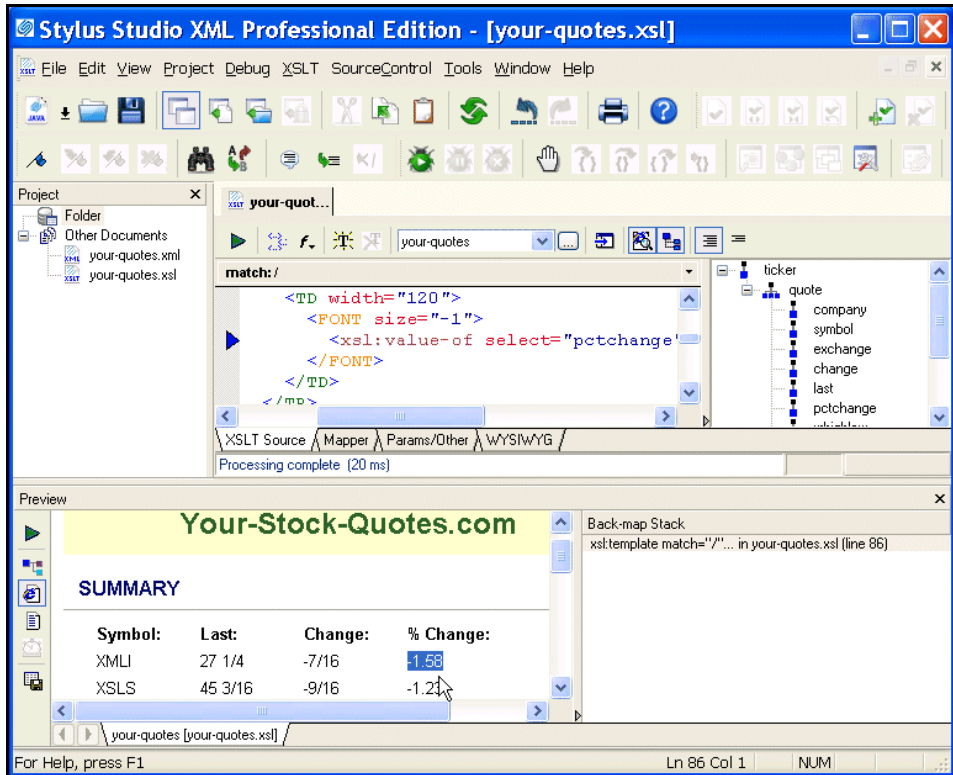
When you preview a result, Stylus Studio automatically saves the changes you have made to the document. If you want to revert to the document's previous state, you can use the undo function (**Edit > Undo**).

## Working with a Sample Result Document


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

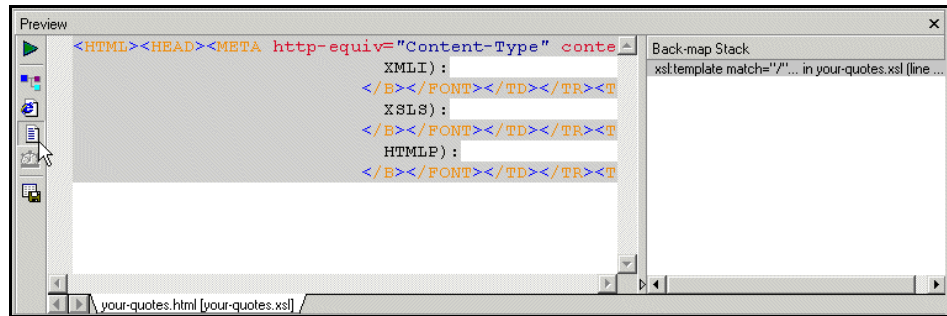
1. In the **Preview** window, click anywhere in the display.

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 29. 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 30. 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. This results in a nicer looking WYSIWYG display.

An alternative to editing the XSLT source is to use the **WYSIWYG** tab to compose HTML graphically. This feature is described in [“Making a Static Web Page Dynamic Using the WYSIWYG Editor”](#) on page 58.

## Importing a Sample HTML File



The HTML to XSLT Document Wizard is available only in Stylus Studio XML Professional Edition.

### ◆ 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 (`<xs1: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**.

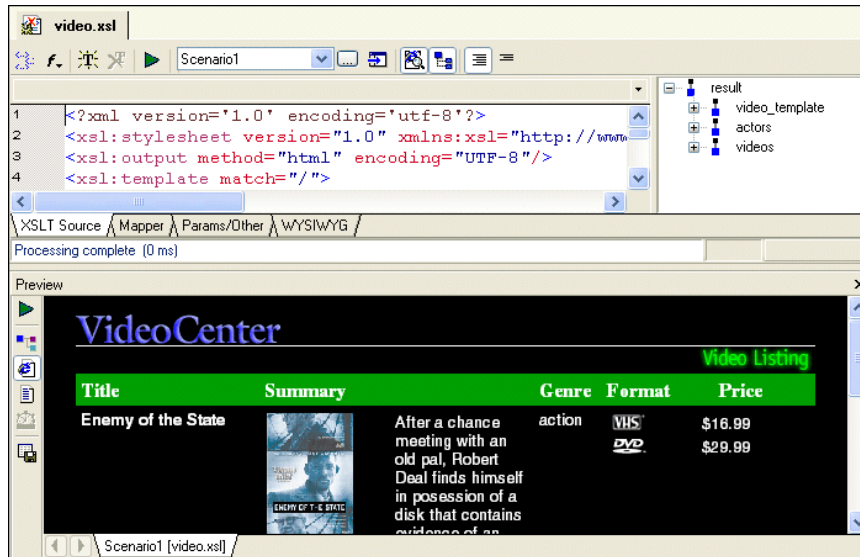
5. In the XSLT editor tool bar, click **Preview Result** . Stylus Studio displays the **Scenario Properties** dialog box. A scenario associates a stylesheet with an XML source document. When Stylus Studio imports an HTML file, it automatically creates the stylesheet that can generate that HTML file.
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 31. 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

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

To modify the stylesheet using the **WYSIWYG** tab, see [“Making Repeating Table Rows in the WYSIWYG Editor”](#) on page 59.

## 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**.

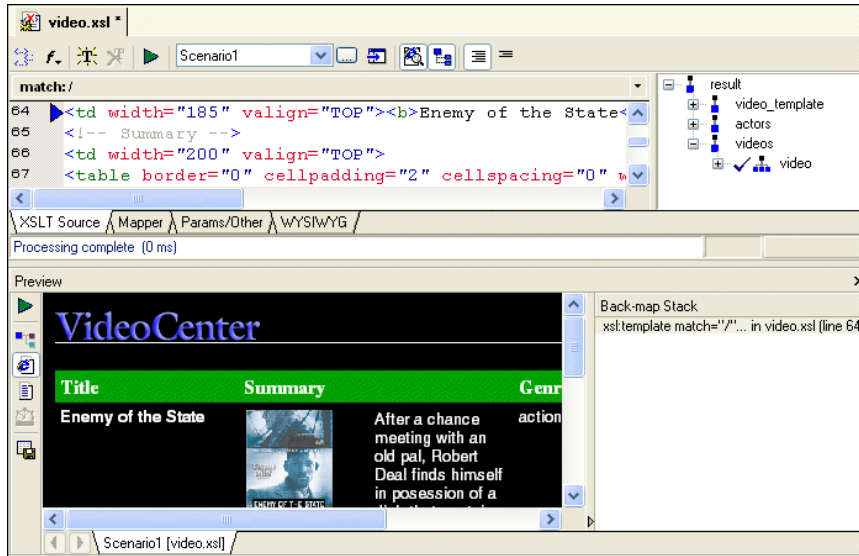
2. Double-click the **video** repeating element.

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.



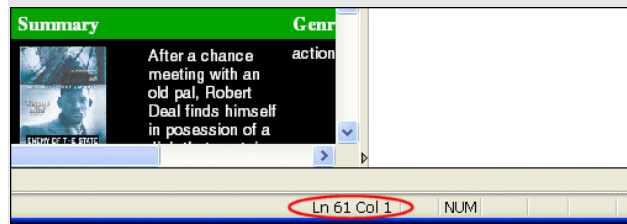
**Figure 32. Back-mapping Shows What Source Generates Output**

**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.

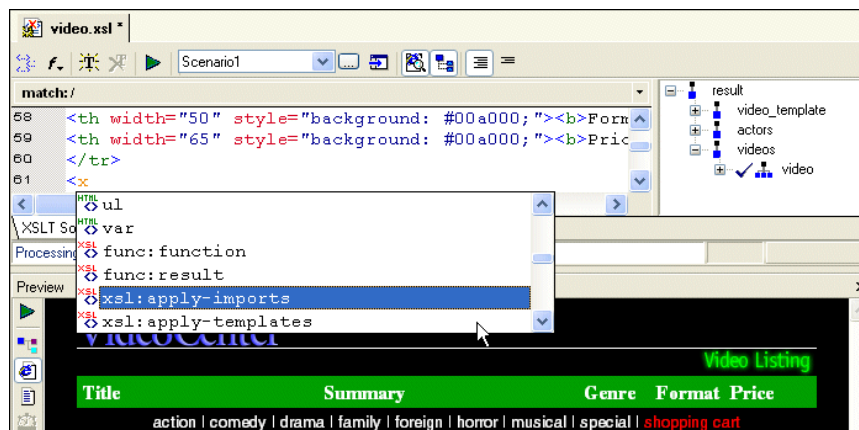


Figure 33. Sense:X Tag Completion in XSLT

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 `<b>` and `</b>` 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 344.

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** .

## Stylesheets That Generate HTML – Getting Started



The XSLT Editor **WYSIWYG** tab is available only in Stylus Studio XML Professional Edition.

You can use the **WYSIWYG** tab of the XSLT Editor to create stylesheets that generate HTML. Stylus Studio automatically generates the XSLT instructions that output the HTML you define on the **WYSIWYG** tab.

This section helps you get started working with stylesheets that generate HTML. To get started with the features common to all stylesheets, see [“Working with Stylesheets – Getting Started”](#) on page 26. To focus on stylesheets that map XML to XML, see [“Using the XSLT Mapper – Getting Started”](#) on page 64.

This section is organized as follows:

- [“Video Demonstrations of the XSLT WYSIWYG Editor”](#) on page 47
- [“Getting Started with the XSLT WYSIWYG Editor”](#) on page 48
- [“Making a Static Web Page Dynamic Using the WYSIWYG Editor”](#) on page 58

Before you begin

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

### Video Demonstrations of the XSLT WYSIWYG Editor

Stylus Studio provides several video demonstrations that show you how to use the XSLT WYSIWYG editor. You can watch these videos instead of (or in addition to) performing the tutorial procedures in this section.

#### Descriptions

Video demonstrations for using the XSLT WYSIWYG editor include the following:

- **Introduction to WYSIWYG in Stylus Studio** – shows how to access the XSLT WYSIWYG editor, how to create static and dynamic contents, and how to create tables and nested tables.
- **Working with Tables** – shows how to sort a table and how to do advanced formatting using XSLT conditional processing on atomic entities or on table rows.

- **Using Templates** – shows how to use the `xs1:apply-templates` instruction with the XSLT WYSIWYG editor, how to create new templates, how to edit templates, how to reuse them.

**Note**

This video demonstration is not described in the tutorial procedures in this section.

- **Converting Static HTML** – shows how to convert a static HTML page to a dynamic HTML page.

The videos are available here: [http://www.StylusStudio.com/xml\\_videos.html](http://www.StylusStudio.com/xml_videos.html)

**Tip**

This page also contains video demonstrations for other Stylus Studio features.

### System Requirements

The system on which you play Stylus Studio video demonstrations must meet the following requirements:

- Windows 2000 or XP operating system.
- Windows Media Player 9 or higher. You can download the Windows Media Player from <http://www.microsoft.com/windows/windowsmedia/download/>

In addition, for best results set your Media Player options as follows:

- Uncheck the **Use Overlays (Tools > Options > Performance > Advanced)**
- Set **Video Size** to 100% (**View > Video Size**)

## Getting Started with the XSLT WYSIWYG Editor

This section provides instructions for getting started with the XSLT WYSIWYG editor. It introduces the basic WYSIWYG editor features, and provides background information needed to edit simple stylesheets.


You should perform the steps in each topic before you move on to the next topic. After the first topic, some steps in subsequent topics depend on actions you performed in a previous topic.

This section is organized as follows:

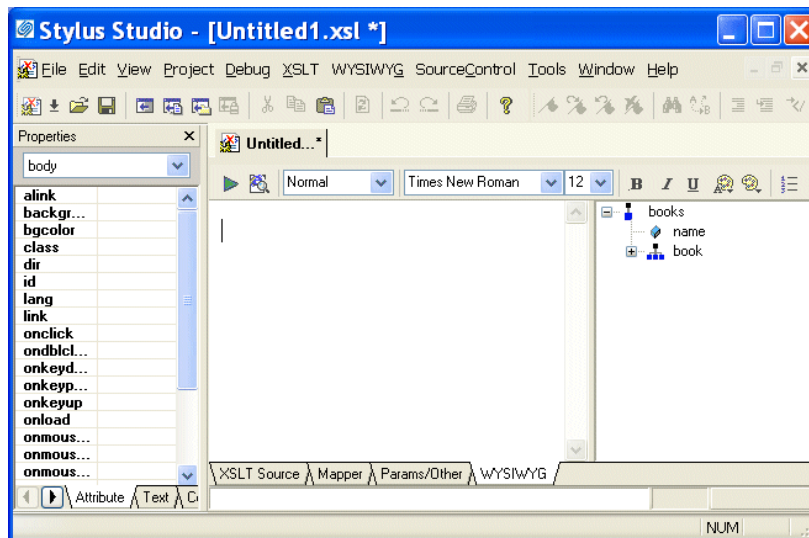
- “[Creating Static HTML](#)” on page 49
- “[Defining Dynamic Contents](#)” on page 52
- “[Adding a Table with Dynamic Contents](#)” on page 53
- “[Using the Properties Window](#)” on page 55

## Creating Static HTML

◆ **To create static HTML:**

1. In the Stylus Studio menu bar, select **File > New > XSLT: WYSIWYG**. Stylus Studio displays the **Scenario Properties** dialog box.
2. In the **Scenario Properties** dialog box, to the right of the **Source XML URL** field, click **Browse** .
3. Navigate to and select `examples\query\books.xml` in the Stylus Studio installation directory, and click **Open**.
4. In the **Scenario Properties** dialog box, click **OK**.

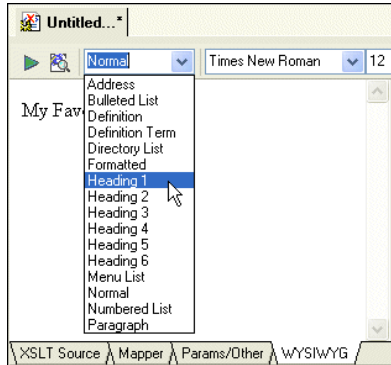
Stylus Studio displays the XSLT Editor open to the **WYSIWYG** tab. The HTML canvas, the center pane on the **WYSIWYG** tab, is empty. A tree that represents the XML source document, `books.xml`, appears to the right of the HTML canvas. The **Properties** window is displayed by default.






**Figure 34. XSLT Editor WYSIWYG Tab**

5. Click the **XSLT Source** tab. As you can see, the basic XSLT instructions that Stylus Studio inserts in every new stylesheet are there. Because you created the stylesheet by selecting the WYSIWYG editor, which is used to design HTML, Stylus Studio has set the output method to HTML (`<xsl:output method="html"/>`).
6. Click the **WYSIWYG** tab.

- Click in the HTML canvas and type My Favorite Books.
- In the **WYSIWYG** tool bar, in the **HTML Element** field, click the down arrow to display the element menu and click **Heading 1**.



**Figure 35. Select HTML Formatting from Drop-down Menus**

- In the HTML canvas, select **My Favorite Books**.
- In the **WYSIWYG** tool bar, click **Text Color**  and select a color, and then click **Center** . You can enter and format text in the HTML canvas as you would in any other WYSIWYG editor. What you are creating is static HTML.
- Click **Preview Result** .  
The **Save As** dialog box appears.
- Type `myBooks.xsl` in the **URL** field and click the **Save** button.
- As you can see in the **Preview** window, the contents of the HTML canvas and the contents of the result document are identical.

**Tip**

Resize the **Preview** window by dragging the splitter that separates the **Preview** window from the XSLT Editor.



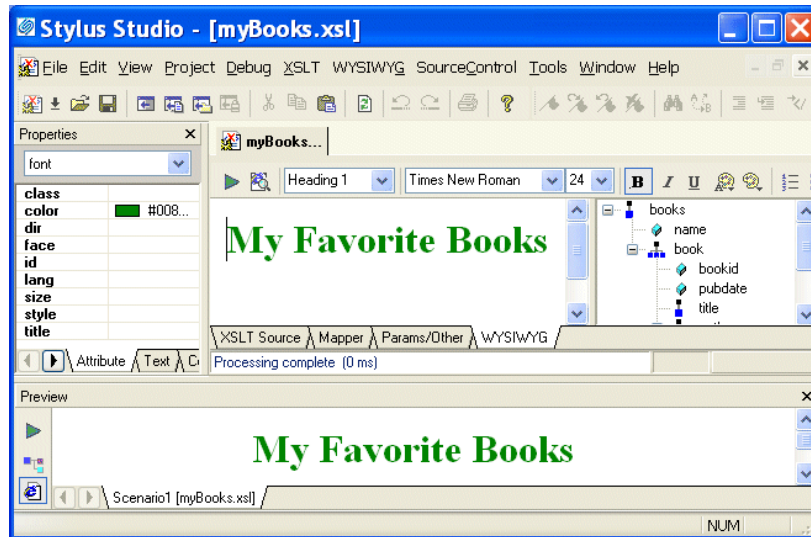


Figure 36. Preview Result in the WYSIWYG Editor

- Click the **XSLT Source** tab. Stylus Studio has captured the formatting and translated it into XSLT instructions. The stylesheet now contains these instructions in the template that matches the root node.

```
<?xml version='1.0' encoding='utf-8'?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>

<xsl:template match="/">
  <p align="center">
    <html><head/>
      <body>
        <font color="#008000">My Favorite Books</font>
      </body>
    </html>
  </p>
</xsl:template>


</xsl:stylesheet>
```

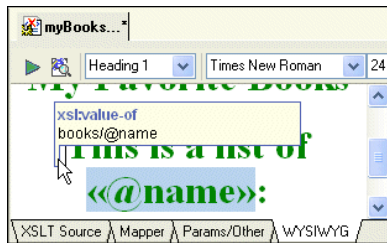
- Click the **WYSIWYG** tab before continuing.

## Defining Dynamic Contents

This topic is part of a sequence that starts with “[Creating Static HTML](#)” on page 49.

◆ **To define dynamic contents:**


1. In the source XML document tree, click books and press the asterisk key (\*) in the numeric keypad to expand all the elements.
2. In the HTML canvas, position the cursor at the end of **My Favorite Books** and press Enter.
3. Type This is a list of :.
4. In the XML source tree, drag the name attribute to just before the colon (:). Stylus Studio displays a pop-up menu.
5. Click **xsl:value-of**. Stylus Studio adds <<@name>> to the canvas; this statement is a placeholder for the value returned by /books/@name.
6. Move the pointer over this placeholder to display a yellow marker . This is the value-of marker.
7. Move the pointer over the value-of marker.  
Stylus Studio displays the instruction that generates the contents for the placeholder.



**Figure 37. xsl:value-of Marker in the HTML Canvas**

In this example, it is xsl:value-of. Stylus Studio also displays the context for the placeholder. In this example, the context is books/@name.


8. Select **This is a list of <<@name>>:**
9. In the WYSIWYG tool bar, click the down arrow in the **HTML Element** field.
10. Click **Heading 2**. As you can see, you can format a mixture of static and dynamic HTML.

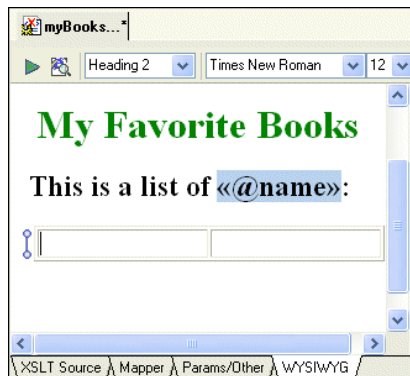
11. Click **Preview Result** . The result document contains **This is a list of My books:** where the HTML canvas contains **This is a list of <<@name>>:**
12. Press the Enter key to move the cursor to the next line before continuing.

## Adding a Table with Dynamic Contents

This topic is part of a sequence that starts with “[Creating Static HTML](#)” on page 49.

### ◆ To add a table with dynamic contents:


1. In the XML source tree, notice the  symbol; this symbol identifies repeating elements, such as book and author.
2. Drag the book element from the XML source tree and drop it onto the HTML canvas. Stylus Studio displays the pop-up menu with an additional option, **Add Table**.
3. Click **Add Table**.  
Stylus Studio inserts a two-column table with a marker to the left of the table. This marker indicates a looping element.



**Figure 38. Table Created Using WYSIWYG Editor**

4. Move the pointer over the marker. Stylus Studio displays the instruction that generates the contents of the table, **xsl:for-each**, and the context for the instruction, **books/book**. Anything you enter in the table is in the context of books/book.
5. Drag the `title` element from the XML source tree and drop it into the first column of the table in the HTML canvas.  
Stylus Studio displays a pop-up menu.

6. Click **xsl:value-of**.  
Stylus Studio displays the «**title**» placeholder in the first column. When you apply the stylesheet, the title of each book will appear in a row of the first column of the table.
7. Drag the author element from the XML source tree and drop it into the second column of the table in the HTML canvas.  
Stylus Studio displays the pop-up menu.
8. Click **Add table**.  
Stylus Studio displays a subtable in the second column of the table.
9. Right-click in the second column of the subtable to display a new pop-up menu.
10. Click **Delete Columns** to delete the second column in the subtable.
11. Drag the author element from the XML source tree and drop it into the subtable again.
12. In the pop-up menu that appears, click **xsl:value-of**.  
Stylus Studio displays the «**author**» placeholder in the second column.

- Click **Preview XSLT** . The result document now contains a table that displays the title and authors for each book in `books.xml`.

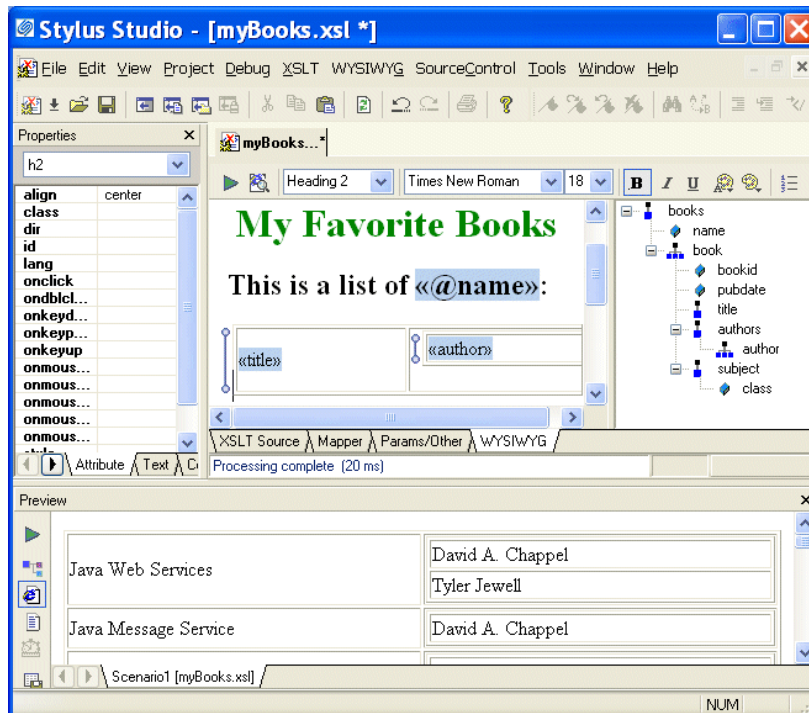


Figure 39. Table with Dynamic Content

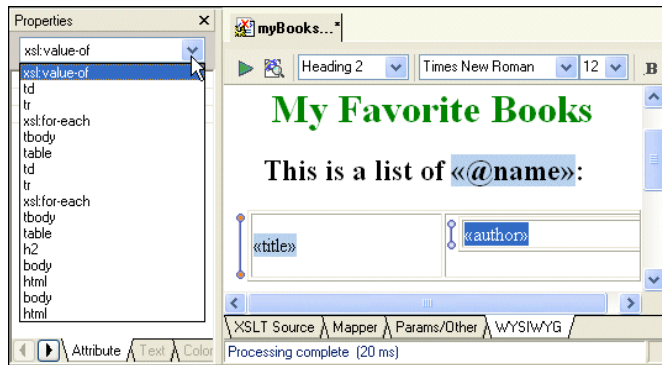
## Using the Properties Window

This topic is part of a sequence that starts with “[Creating Static HTML](#)” on page 49.

◆ **To get started using the HTML editor to format a table with dynamic contents:**

- In the HTML canvas, click the `«author»` placeholder, which represents an `xs1:value-of` instruction.

2. In the **Properties** window, click the down arrow to display a list of elements.



**Figure 40. Drop-Down List Shows Enclosing Elements**

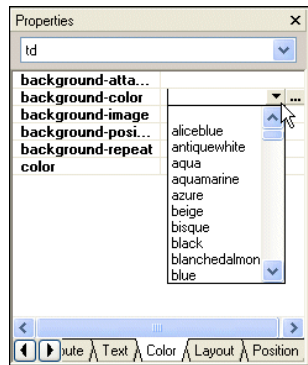
Starting with the `xsl:value-of` element itself, these are the elements that enclose the `xsl:value-of` element in the XSLT, from the innermost to the outermost. You can specify properties for any of these elements, and what you specify can affect the display represented by the placeholder you clicked.

3. Click the second `td` element to specify properties for the table data. (If you click the first `td` element, you specify properties for the subtable that is in the second column of the main table.)


Stylus Studio displays only the properties that can be specified for the element you clicked. For the `td` element, there are many such properties, or attributes, and you can click the tabs at the bottom of the **Properties** window to view a particular subset (**Color**, **Layout**, **Position**, and so on).

4. At the bottom of the **Properties** window, click the **Color** tab.
5. Double-click the **background-color** field.

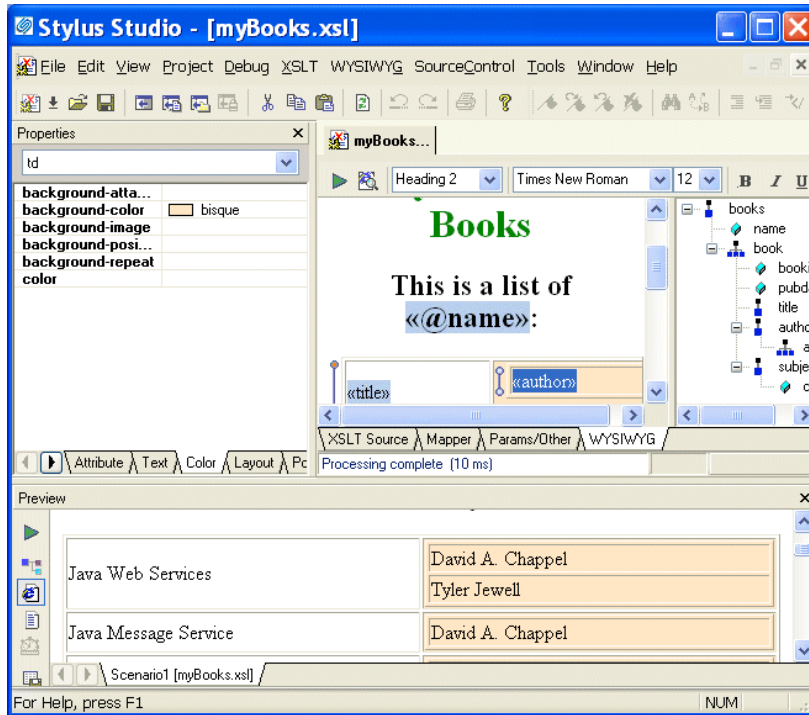
- Click **Browse**  to display a drop-down color menu.



**Figure 41. WYSIWYG background-color Drop-Down Menu**

- Click the color you want for the background of the table rows and click OK.
- Click **Preview Result** .

The result document now displays the table rows with the background color you selected.



**Figure 42. Colors Applied by Editing Properties**

9. Click the **XSLT Source** tab.

Stylus Studio has inserted the XSLT instructions that display the table in the color you specified.

## Making a Static Web Page Dynamic Using the WYSIWYG Editor

In “[Making a Static Web Page Dynamic by Editing XSLT](#)” on page 38, you learned how to use the XSLT Editor’s **XSLT Source** tab to build a Web page that displays dynamic content.

This section provides step-by-step instructions for using the Stylus Studio XSLT WYSIWYG editor to convert a static Web page that displays information about one video into a dynamic Web page that displays information about many videos.



## Before You Begin

Before starting this part of the tutorial, you must import an HTML file as described in “[Importing a Sample HTML File](#)” on page 39, with the following changes:

- In [Step 6](#), give the scenario a different name, or use the current default.
- In [Step 11](#), give the stylesheet a different name.

When you are done, your copy of Stylus Studio should look something like this:

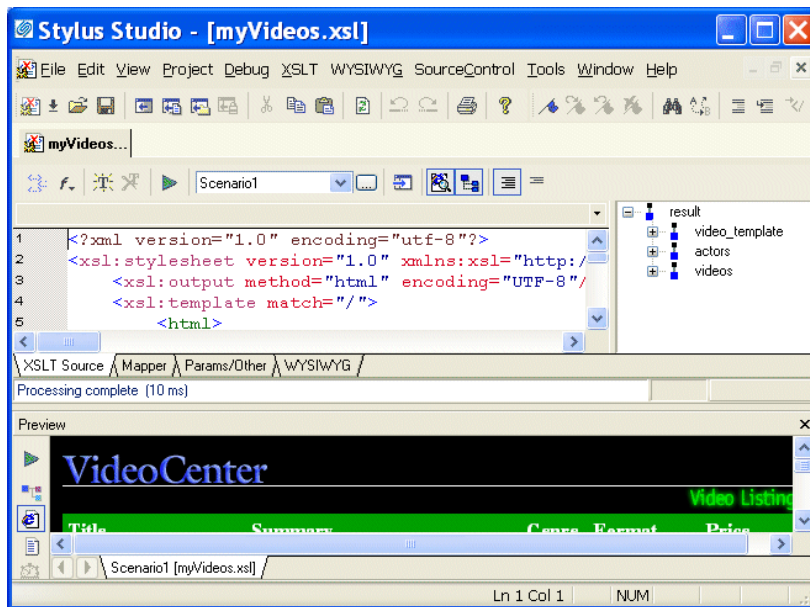


Figure 43. Preview of myVideos.xsl

After you perform [Step 11](#) in that topic, return here, and start with “[Making Repeating Table Rows in the WYSIWYG Editor](#)” on page 59.

## Making Repeating Table Rows in the WYSIWYG Editor

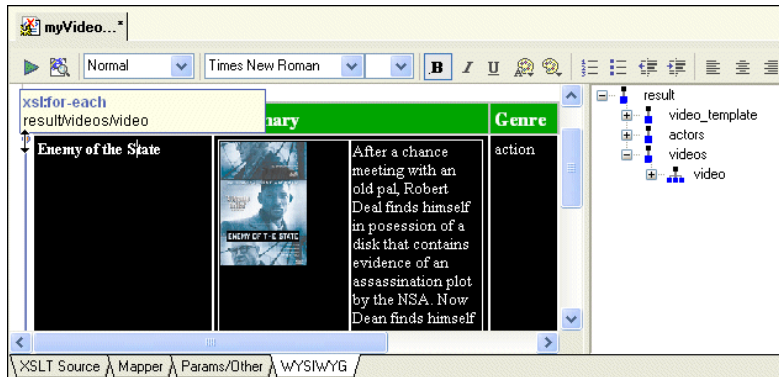
This topic is part of a sequence that starts with “[Importing a Sample HTML File](#)” on page 39.

In the **Preview** window, you can see that the information for one video appears in a row of a table. You want to make this a repeating row and have each row contain information for a different video.


◆ **To create repeating rows in the HTML canvas:**

1. Click the **WYSIWYG** tab.
2. In the source tree pane, expand the **result** and **videos** nodes.
3. Drag the video repeating element to the *Title* table cell, which contains *Enemy of the State*.
4. In the pop-up menu that appears, click **Make repeating**.

Stylus Studio displays a looping marker to the left of the table row to indicate that it is a repeating row. If you move the cursor over the marker, Stylus Studio displays the information that the marker represents an `xs1:for-each` instruction that executes in the context of `/result/videos/video`.



**Figure 44. Repeating Element Depicted in WYSIWYG**

5. Click **Preview Result** . The result document now displays multiple rows. Each row contains information for the same movie: *Enemy of the State*.



Now that the XSLT describes a table with repeating rows, our next task is to make each of the rows' columns dynamic, that is, to ensure that each row displays information for a different movie.

## Making Contents Dynamic in the WYSIWYG Editor

This topic is part of a sequence that starts with [“Importing a Sample HTML File”](#) on page 39.

◆ **To create dynamic contents in the HTML canvas:**

1. In the HTML canvas, select the title *Enemy of the State*.

- In the source tree pane, expand the **video** node and drag and drop the **title** element on the selected text.
- In the pop-up menu that appears, click **xsl:value-of**.  
Stylus Studio removes the text for *Enemy of the State* and replaces it with the `<<title>>` placeholder.
- Move the cursor over the `<<title>>` placeholder and then shift it slowly over the value-of marker (`|`) to the left.  
Stylus Studio displays the information that the marker represents an `xsl:value-of` instruction and that it executes in the context of `/result/videos/video/title`.
- Click **Preview Result** .  
The result document now displays a different video title in each row. If you want, you can select the title placeholder, apply some formatting and preview the results. Every title in the result document will have the formatting you specified.
- Repeat [Step 1](#) through [Step 4](#) for each of the following elements:
  - summary - select all the text in the summary before you drag the **summary** element
  - genre - select *action*
  - vhs - select *16.99* (leave the dollar sign (\$))
  - dvd - select *29.99*
  - director - select *Tony Scott*
  - studio - select *Buena Vista*
  - year - select *1999*
  - runtime - select *113*
- Click **Preview Result** . Each row now contains information for a different video. However, the video image in each row is still the same.

### Rendering Images as Dynamic Content in the WYSIWYG Editor


This topic is part of a sequence that starts with [“Importing a Sample HTML File”](#) on page 39.

The image for each movie is still the same. For this part of the XSLT, you will use the **XSLT Source** tab:

- Click the **XSLT Source** tab.

2. In the **Preview** window, click the video image to backmap to the instruction that generates it in the **XSLT Source** tab.

Stylus Studio displays the XSLT Source tab, and flags the line that contains the instruction. As you can see, the stylesheet uses an image ID.

3. In the string "images/video/id1244100.gif", select id1244100.
4. Type {@id} to replace the selected text. This specifies an attribute value template.
5. Click **Preview Result** .

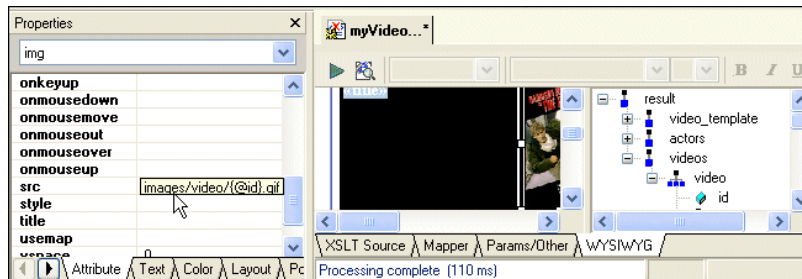
Each row now contains the correct image for the video described in that row.

6. Click the **WYSIWYG** tab.

As you can see, there is now a placeholder where the original image was.


7. Click the image's placeholder.

In the **Properties** window, Stylus Studio displays the properties for the image. If you scroll down to the **src** property, you can see that the value is **images/video/{@id}.gif**.



**Figure 45. XSLT Properties for Image Source**

8. Scroll the result document towards the bottom and examine the VHS and DVD prices. As you can see, some of the prices are missing the decimal places.
9. In the result document, click on the price for a VHS video. Stylus Studio backmaps to `<xsl:value-of select="vhs"/>` in the **XSLT Source** tab.
10. Replace "vhs" with "format-number(vhs, '##.00')". This ensures that every VHS price is formatted with two decimal places.
11. In the **XSLT Source** tab, scroll down about eight lines, and repeat the previous step for the DVD price. That is replace `select="dvd"` with `select="format-number(dvd, '##.00')`.

12. Click **Preview Result** . Each price now includes two decimal places. However, small circles appear when no VHS tapes or DVDs are available for sale. The next topic describes how to use conditional processing to remove these circles.
13. Click the **WYSIWYG** tab. Stylus Studio displays `<<format-number(>>` as the placeholder for the VHS and DVD prices.

## Using Conditional If Processing in the WYSIWYG Editor

This topic is part of a sequence that starts with [“Importing a Sample HTML File”](#) on page 39.


In the current videos result document, there are small circles when no VHS tapes or DVDs are available.



**Figure 46. Circles Show Processing of Empty Strings**

These circles appear when the XSLT processor tries to process an empty string with `select=format-number`.

### ◆ To remove these small circles:

1. In the **WYSIWYG** tab, select the `<<format-number(>>` that is the placeholder for the VHS price.
2. Right-click the selected placeholder.
3. In the pop-up menu, select **Conditional Processing > Add If**. Stylus Studio displays the **Input** dialog box.
4. In the **Input** dialog box, type `vhs` in the **Test Condition** field. This tests whether there are any `vhs` elements.
5. Repeat [Step 1](#) through [Step 4](#), but this time select the DVD price placeholder, and specify `dvd` as the test condition.
6. Click **Preview Result** . The small circles no longer appear. Instead, nothing appears when there are no VHS tapes or DVDs for a particular video.

## Using the XSLT Mapper – Getting Started

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.

How this section is organized

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 64
- [“Mapping Nodes in Sample Files”](#) on page 66
- [“Saving the Stylesheet and Previewing the Result”](#) on page 69
- [“Deleting Links in Sample Files”](#) on page 70
- [“Defining Additional Processing in Sample Files”](#) on page 71

Other sources of information

In addition to the topics described in this section, the *Stylus Studio® 6 User Guide* contains other sources of information on XSLT:

- To learn more about XSLT, see [“Working with XSLT”](#) on page 315.
- To get started XSLT Editor features for stylesheets, see [“Working with Stylesheets – Getting Started”](#) on page 26.
- To focus on features for stylesheets that generate HTML, see [“Stylesheets That Generate HTML – Getting Started”](#) on page 47.
- To learn about the XSLT mapper in greater detail, see [“Creating XSLT Using the XSLT Mapper”](#) on page 439.

Before you begin

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.

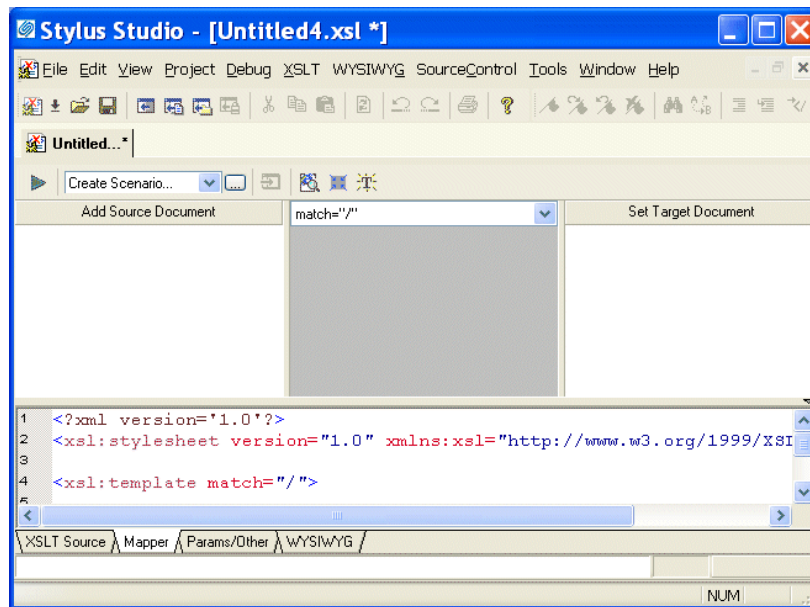
### Opening the XSLT Mapper

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: Mapper**.

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 47. 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.

2. Click the **Add Source Document** button at the top of the mapper's left pane. Stylus Studio displays the **Open** dialog box.
3. For this example, navigate to the `examples\simpleMappings` directory in the Stylus Studio installation directory.
4. Double-click **books.xml**.

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

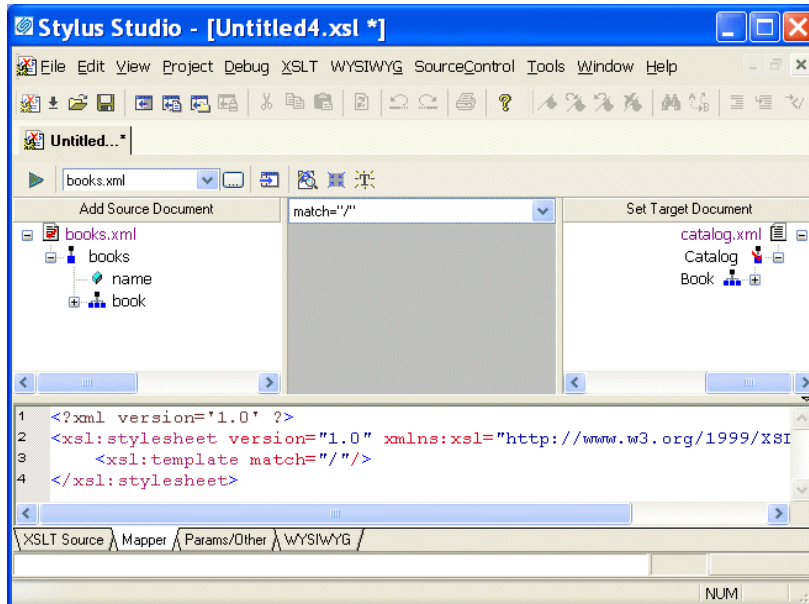


Figure 48. XSLT Mapper Tab with Source and Target Documents

## 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.



- Press and hold the left mouse button, and drag from book to the Book repeating element in catalog.xml.

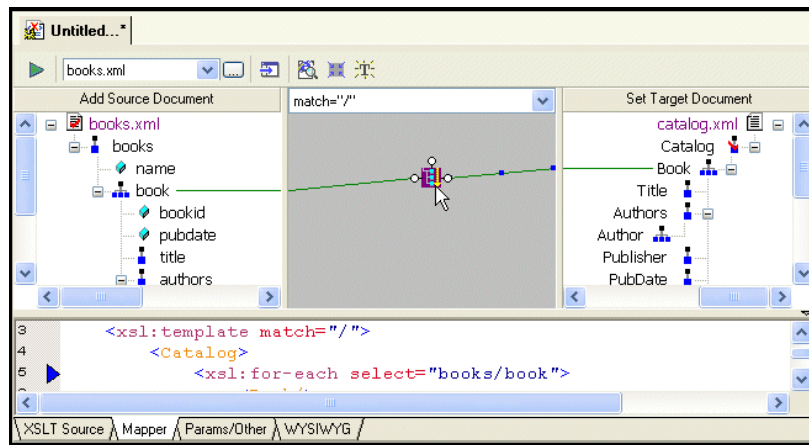
Stylus Studio draws a line as you drag.

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

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

In the **Output method:** field, display the drop-down list and click **xml**. 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 `xsl:output` instruction is added to the XSLT source:

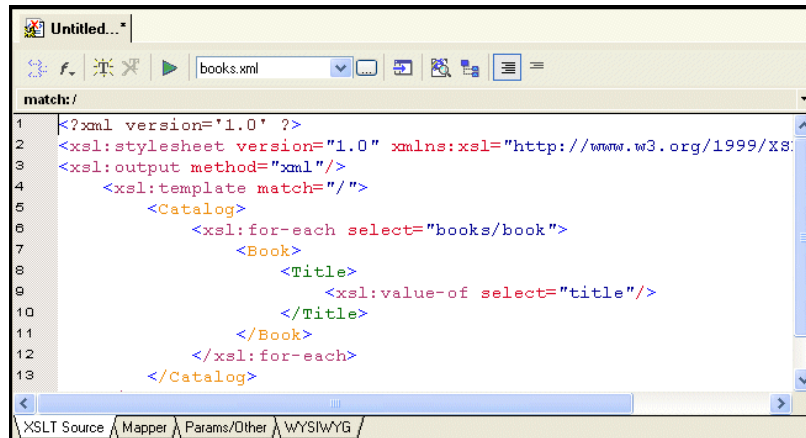
```
<xsl:output method="html"/>
```

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.

8. 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.)




**Figure 50. 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 `xs1: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.xsl`.
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 33.

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



Figure 51. Result of Applying XSLT to `books.xml`

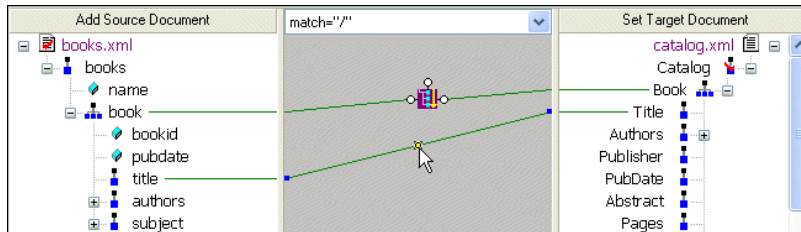
The result document uses the same schema as the target document, `catalog.xml` in this example. Because not all nodes have been mapped yet, the result document does not contain all nodes found in `books.xml` (author and subject nodes, for example).

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.xsl** 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 52. 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 464.

## 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 479.

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 489.

This section includes the following topics:

- “Setting Up Stylus Studio to Debug Sample Files” on page 72
- “Inserting a Breakpoint in the Sample Stylesheet” on page 73
- “Gathering Debug Information About the Sample Files” on page 75

Before you begin

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.

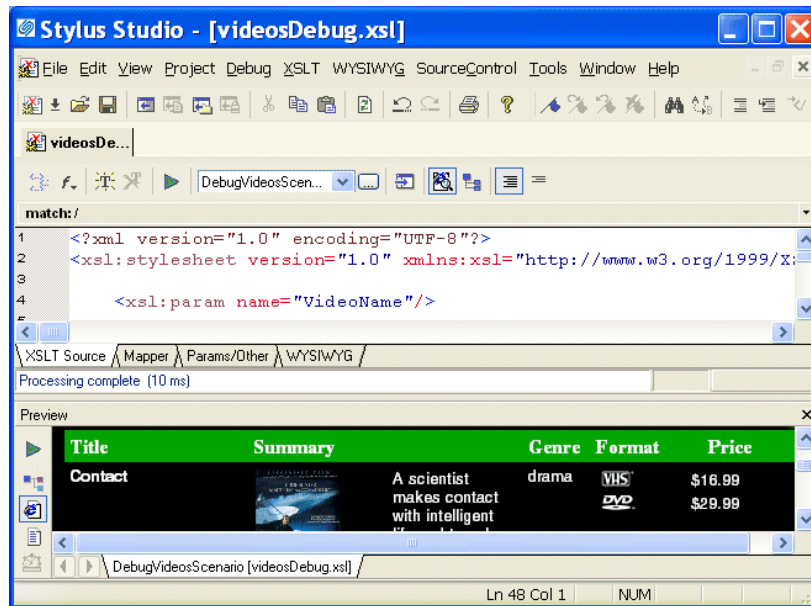


Figure 53. Preview of videosDebug.xsl

## 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">`

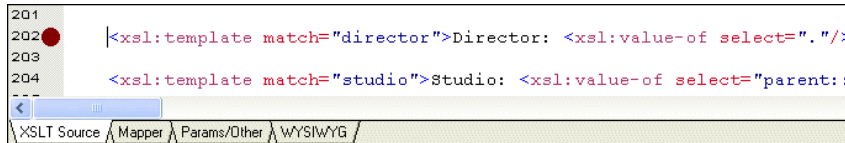
### Tip

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.



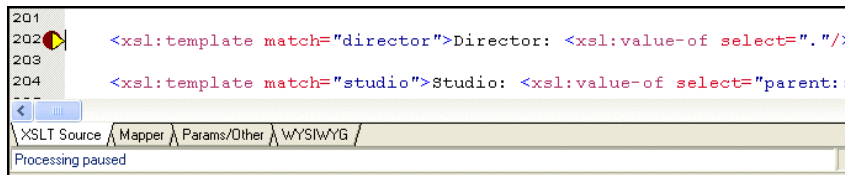
**Figure 54. A Red Circle Shows Where Breakpoints Are Set**

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 55. Yellow Triangle Shows Where XSLT Processing Stopped**

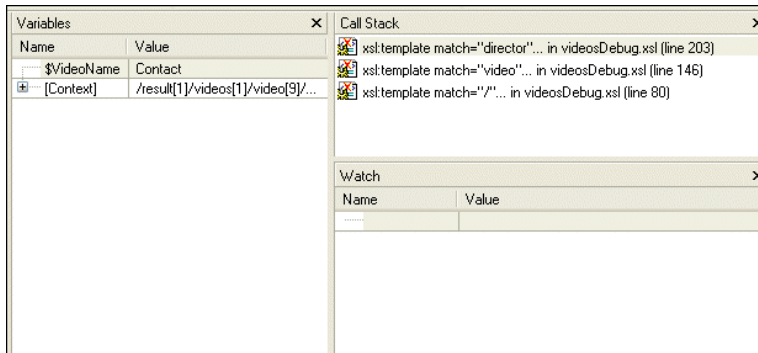
Do not do it, but to stop debugging, you can click **Cancel** in the lower right corner of the XSLT editor window, or click **Stop Debugging**  in the Stylus Studio tool bar.



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

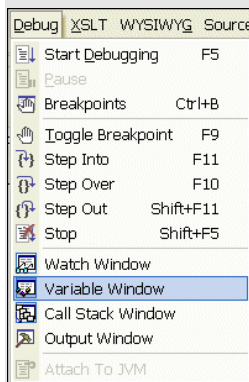
When XSLT processing is suspended at a breakpoint, Stylus Studio displays the **Variables**, **Call Stack**, and **Watch** windows.



**Figure 56. Variable, Call Stack, and Watch Windows Appear During Debugging**

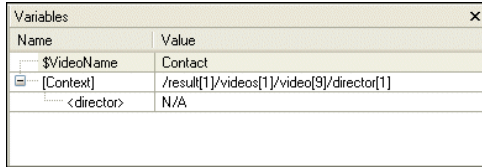
You can use the information in these windows to learn about potential and actual problems encountered in your XSLT processing.

**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.

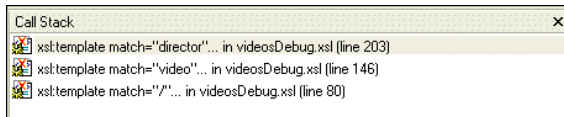


**Figure 57. Variables Window**

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.

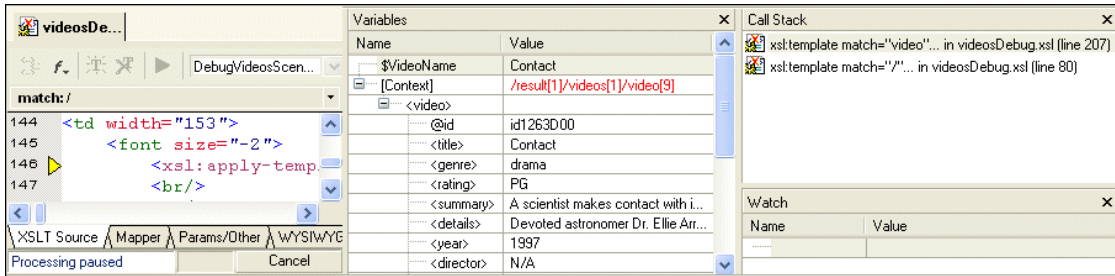


**Figure 58. Call Stack Window**

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.

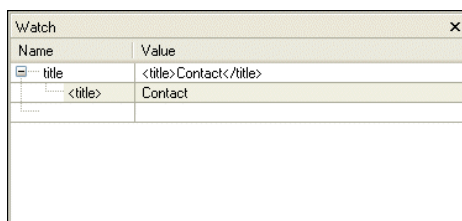


**Figure 59. Stepping Out Advances the Processor**

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 60. Watch Window Lets You Track Variables**

- ◆ **To enter a variable to watch:**

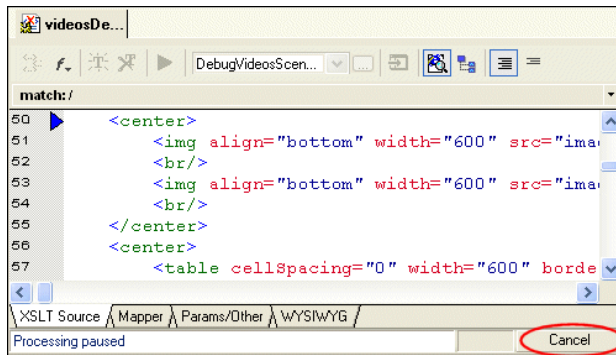
1. Double-click the **Name** field.
2. Type the name of the variable you want to watch and press Enter.

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

## Ending Processing During a Debug Session

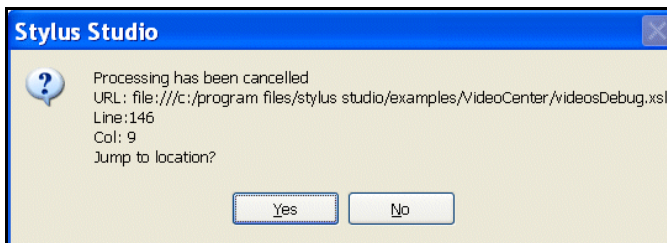
### ◆ To end processing during a debug session:

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 61. 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.

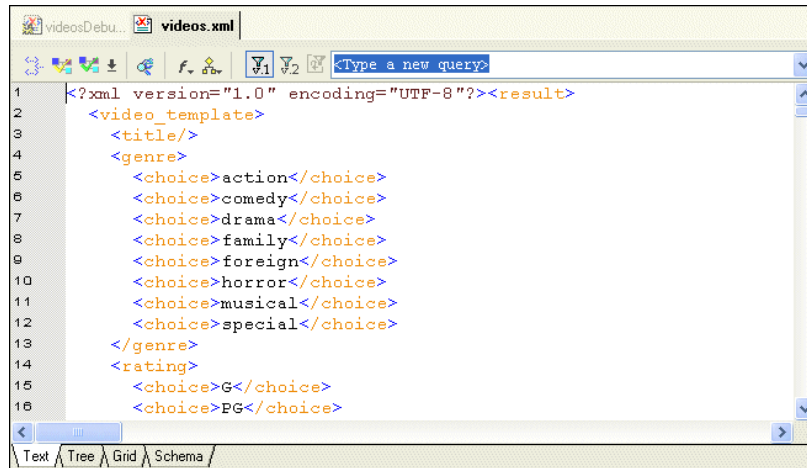


**Figure 62. You Can Jump to Where XSLT Processing Ended**

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 63. Viewing the XML Source for an XSLT Transformation**

For more  
information

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 479.

## 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 589.

### 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 80
- [“Defining Data Elements in a Sample DTD”](#) on page 81
- [“Defining the Container Element in a Sample DTD”](#) on page 82
- [“Defining Structure Rules in a Sample DTD”](#) on page 82
- [“Examining the Tree of a Sample DTD”](#) on page 84

Before you begin

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.

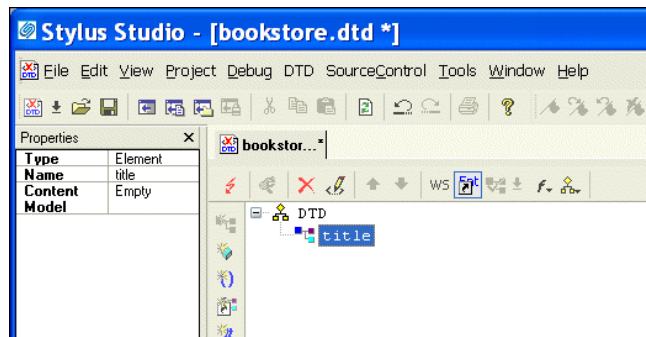



Figure 64. New Element in the DTD Editor

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:

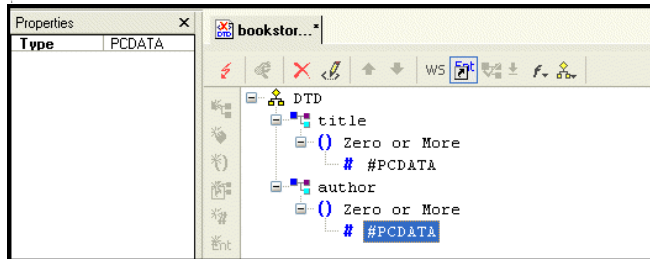





Figure 65. Creating a DTD with Two Elements

## 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

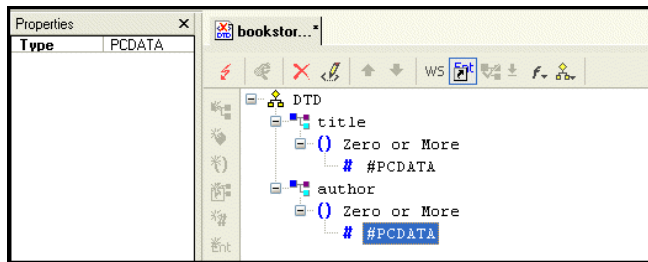
- ◆ **To specify the rules for the structure of the book element:**
  1. Click the **book** node in the DTD tree if it is not already selected.
  2. Click **New Modifier** .
  3. In the drop-down list that appears, scroll down and double-click **Sequence**. This indicates that the book element can include 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 66. Early Steps of bookstore.dtd**

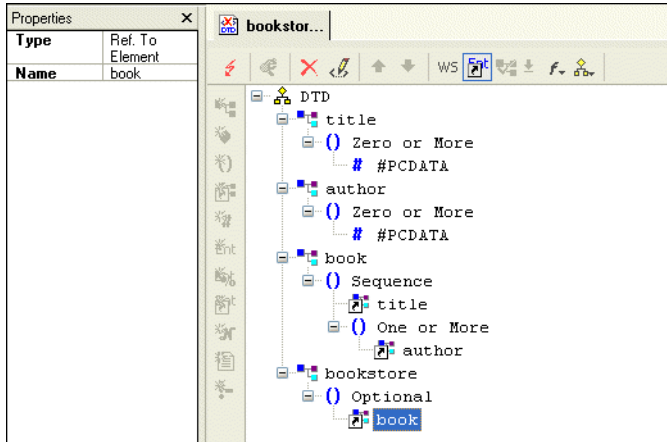
However, you have not yet specified that you want the book element itself to be optional. You need to do this in the element that references the book element. For example, suppose the bookstore element is the root element in XML documents that use this DTD. Further suppose that you want the book element 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 67](#).



**Figure 67. 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 498.

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:

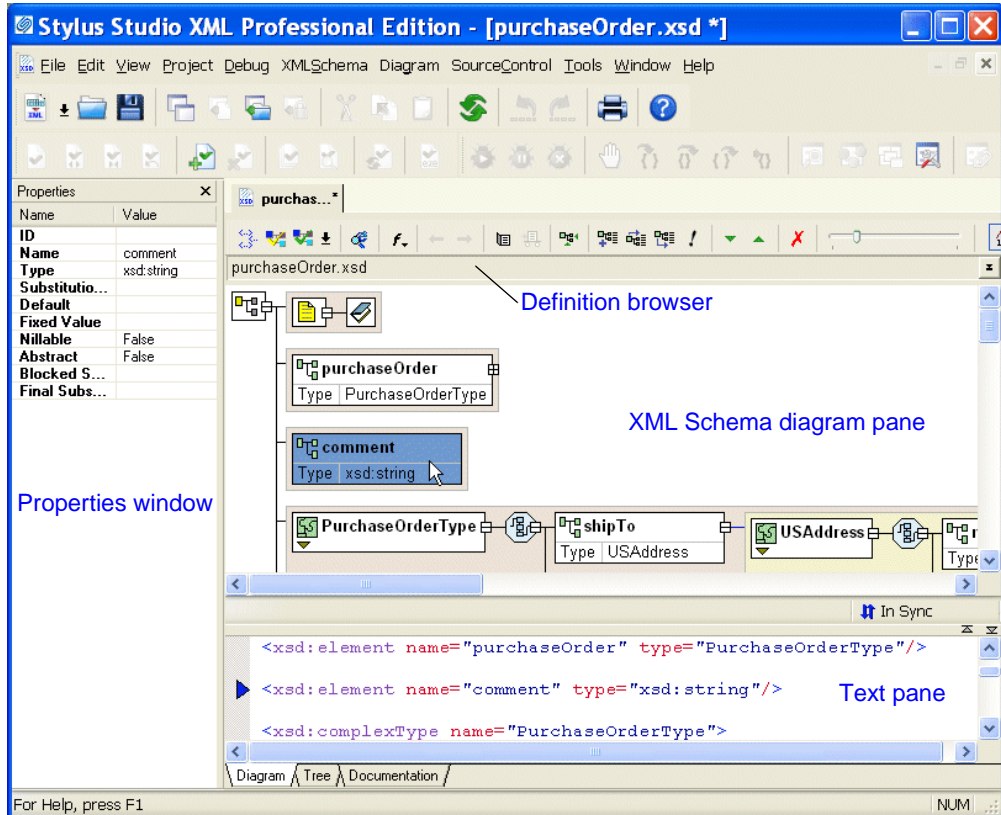
- [“Introduction to the Diagram Tab”](#) on page 86
- [“Editing Tools of the Diagram Tab”](#) on page 95
- [“Description of Sample XML Schema”](#) on page 100
- [“Defining a complexType in a Sample XML Schema in the Diagram View”](#) on page 100
- [“Defining Elements of the Sample complexType in the Diagram View”](#) on page 108

Before you begin

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 68](#).



**Figure 68. 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.

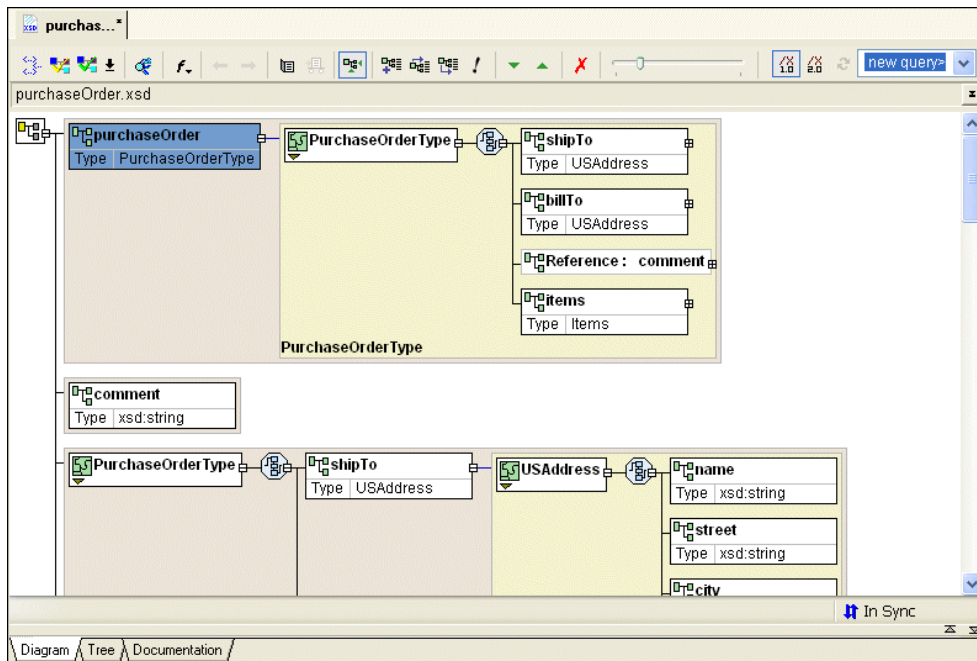
The **Diagram** tab, shown in [Figure 68](#), consists of three main areas:

- “[Diagram Pane](#)” on page 87
- “[Text Pane](#)” on page 92
- “[Definition Browser](#)” on page 94

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 69. 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](#).

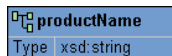
**Table 1. Symbols Used in the XML Schema Diagram**

<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 72](#) for example, the `PurchaseOrderType` complexType has been expanded. The `shipTo` element has not.

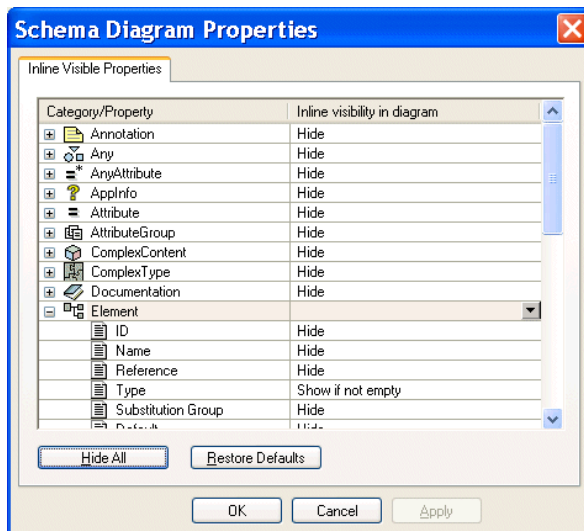
### Displaying Properties

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 70](#).



**Figure 70. 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 71](#). (In addition, the **Properties** window displays all the properties for any node you select.)



**Figure 71. 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**.

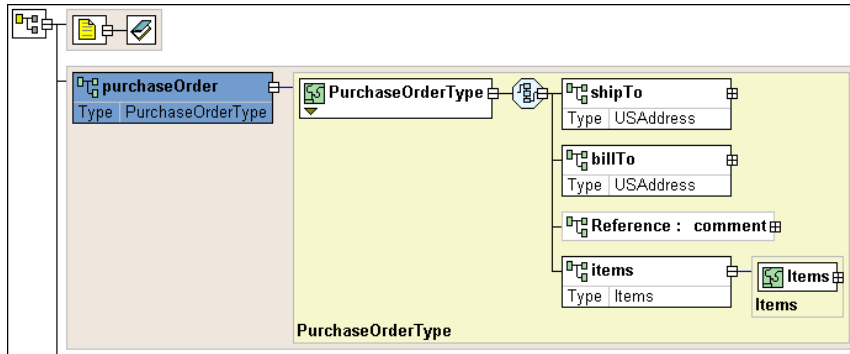
### **Background color**

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 72](#), the `purchaseOrder` element is an example of a globally defined node.



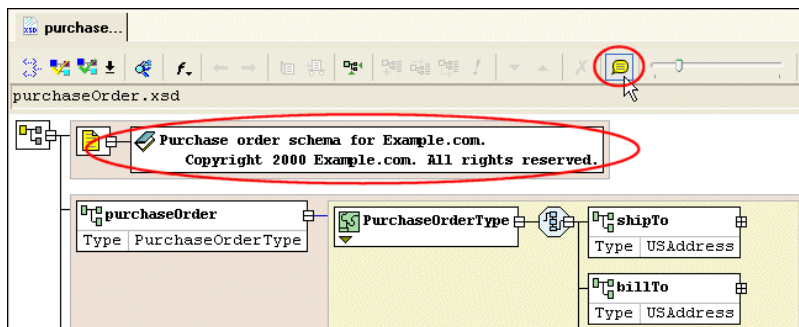
- A light yellow background identifies local instances of globally defined types. In [Figure 72](#), the PurchaseOrderType complexType is a local instance of that type.



**Figure 72. 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 73](#).





**Figure 73. Show Documentation with the Click of a Button**


### Moving around the diagram


There are several ways to move around the diagram pane:

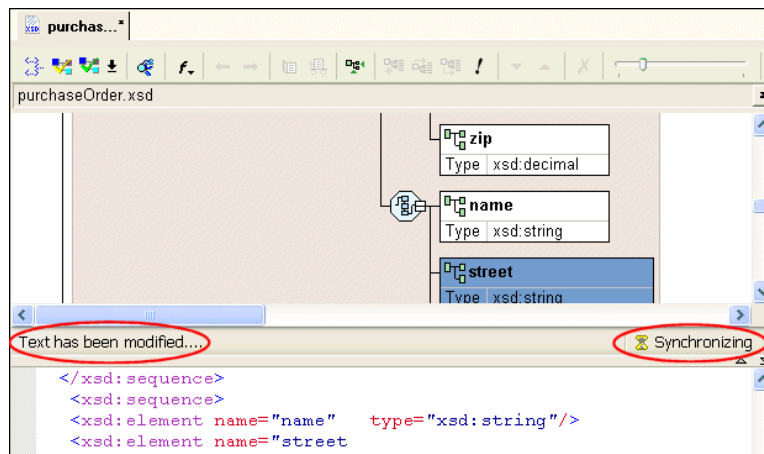
- To move from node to node in the diagram, press the arrow keys.

- 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

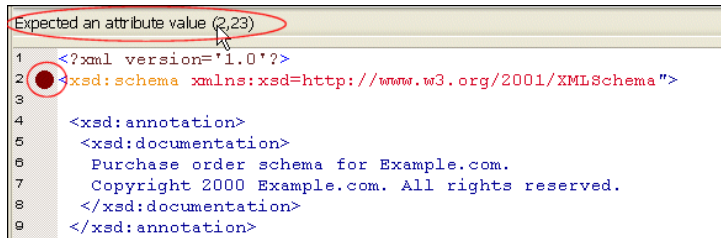
The text pane appears directly beneath the XML Schema diagram pane. It displays the XML Schema code represented by the nodes you create in the diagram. The default font is Courier New, but you can change it to whatever font you want by clicking the **Change Font** button (  ).

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 74](#).



**Figure 74. Synchronization Status is Monitored and Displayed**

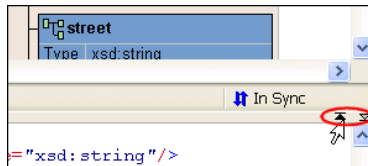
Stylus Studio also flags any XML Schema errors in the text pane – lines that contain errors are identified with a red dot, and the type and location of the error is displayed in the status area at the top of the text pane, as shown here:



**Figure 75. 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.

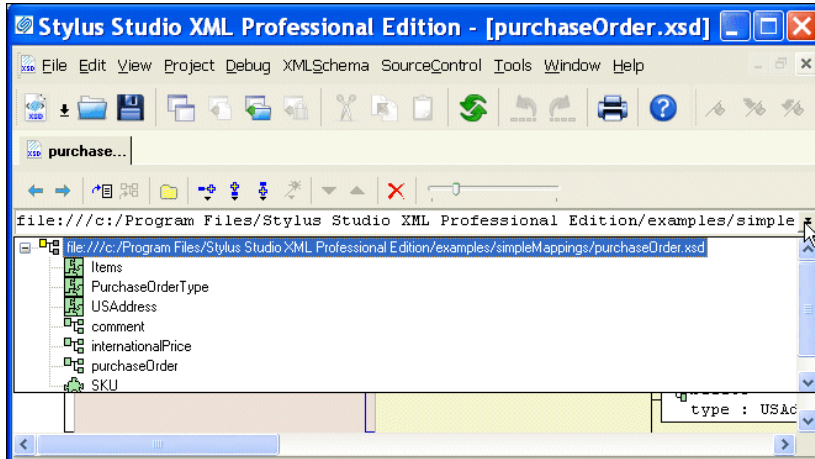


**Figure 76. Splitter Controls Change Size of Text and Diagram Panes**

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.

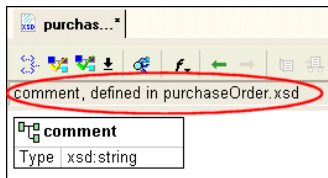
## Definition Browser

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 77. 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 78. 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 95
- “[In-place Editing](#)” on page 95
- “[Drag-and-Drop](#)” on page 96
- “[QuickEdit](#)” on page 97
- “[Refactoring](#)” on page 97

### 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

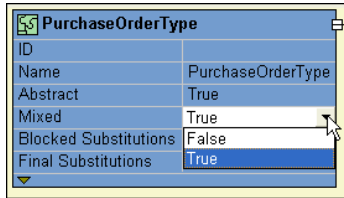
- Select **XML Schema > Diagram > Add > Element** from the main menu
- Select **Add > Element** from the sequence shortcut menu
- Click the **Add** button  on the tool bar and select **Element** from the drop-down list it displays

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 79](#).



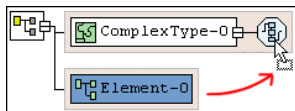
**Figure 79. In-Place Editing**

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 89.

## 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 80](#).



**Figure 80. 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

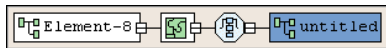
**Tip** Any node you drag to the schema node is created as a child of the schema node.

### 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.



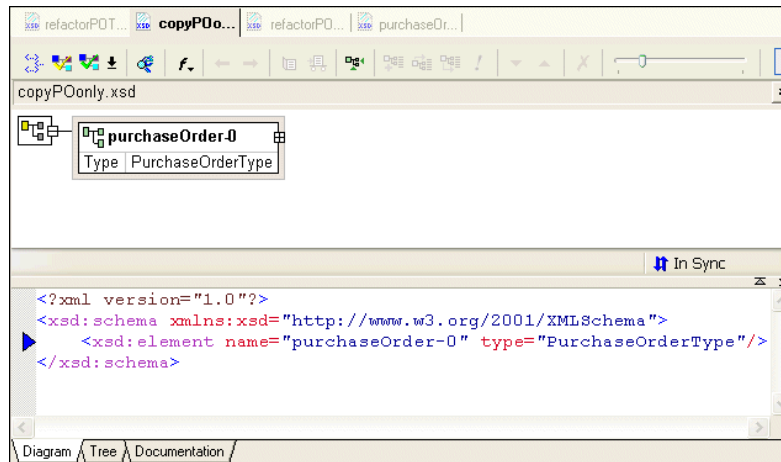
**Figure 81. QuickEdit Creates Complex Definitions With a Click**

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

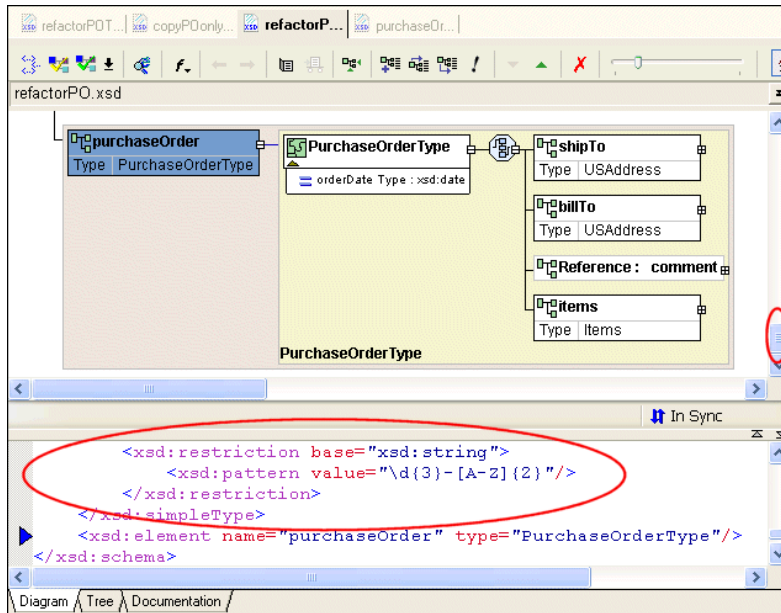
appears when it is *copied* from purchaseOrder.xsd and pasted into a new XML Schema document:



**Figure 82. 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 83. 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 100 are presented in the following topics:

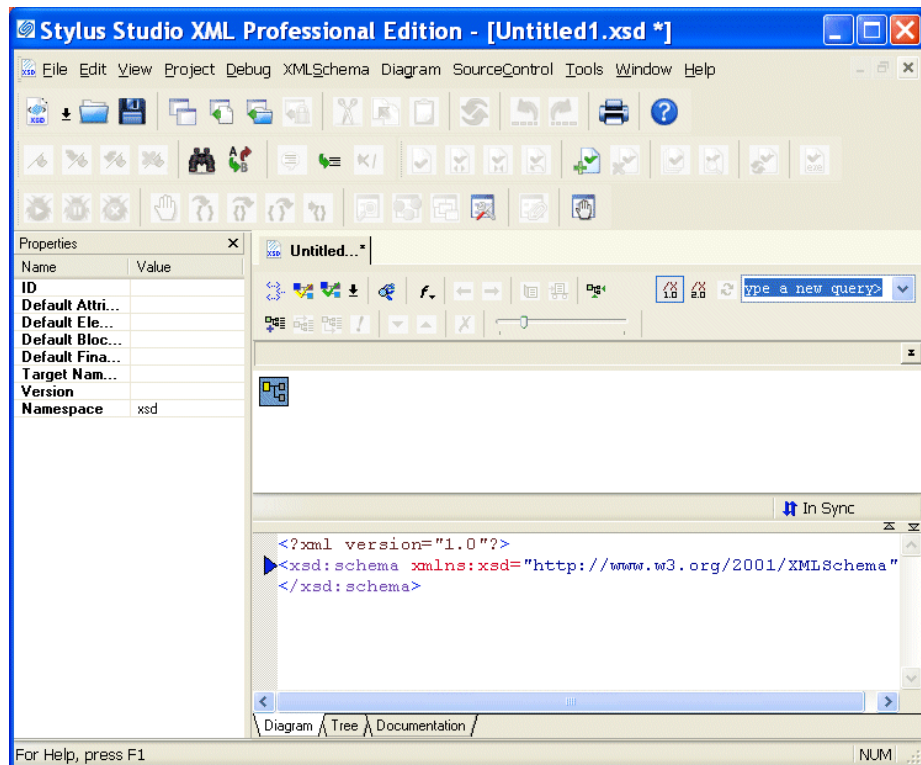
- “[Defining the Name of a Sample complexType in the Diagram View](#)” on page 101
- “[Adding an Attribute to a Sample complexType in the Diagram View](#)” on page 102
- “[Adding Elements to a Sample complexType in the Diagram View](#)” on page 103
- “[Adding Optional Elements to a Sample complexType in the Diagram View](#)” on page 104
- “[Adding an Element That Contains Subelements to a complexType in the Diagram View](#)” on page 104

- “Choosing the Element to Include in a Sample complexType in the Diagram View” on page 106

### Defining the Name of a Sample complexType in the Diagram View

#### ◆ To define a complexType in a sample XML Schema:

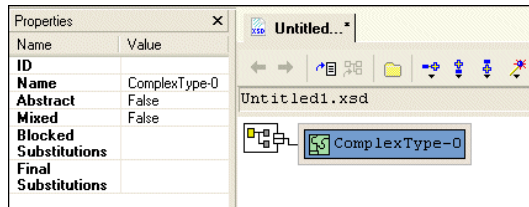
1. From the Stylus Studio menu bar, select **File > New > XML Schema**.  
Stylus Studio displays the XML Schema Editor. Maximize the XML Schema Editor window. If the **Project** window is visible, you can close it.
2. At the bottom of the XML Schema editor, click the **Diagram** tab.  
Stylus Studio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




**Figure 84. 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.

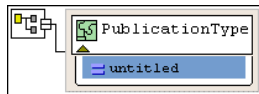


**Figure 85. New complexType**

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

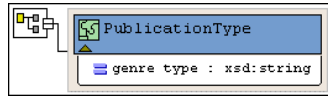
- ◆ **To add the genre attribute to the PublicationType complexType:**
  1. Right-click the `PublicationType` node.
  2. In the shortcut menu that appears, select **Add > Attribute**.  
Stylus Studio displays a node for the new attribute (`untitled`).



**Figure 86. 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 87](#).



**Figure 87. 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)

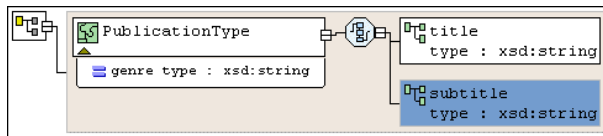
and Max Occur. (maximum occurrences) properties is 1. You want exactly one instance of the `title` element in `PublicationType`, so you can accept these defaults.

### 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 88](#):




**Figure 88. `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 100) 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.

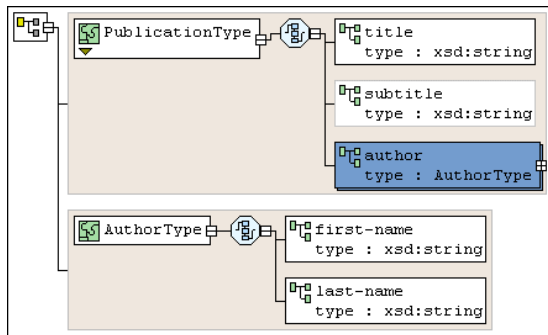
**Tip** A plus sign appears on the right side of the author element. You can click the plus sign to display the definition of the AuthorType complexType, which was just added to the author element.

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 89](#):



**Figure 89.** 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 90:

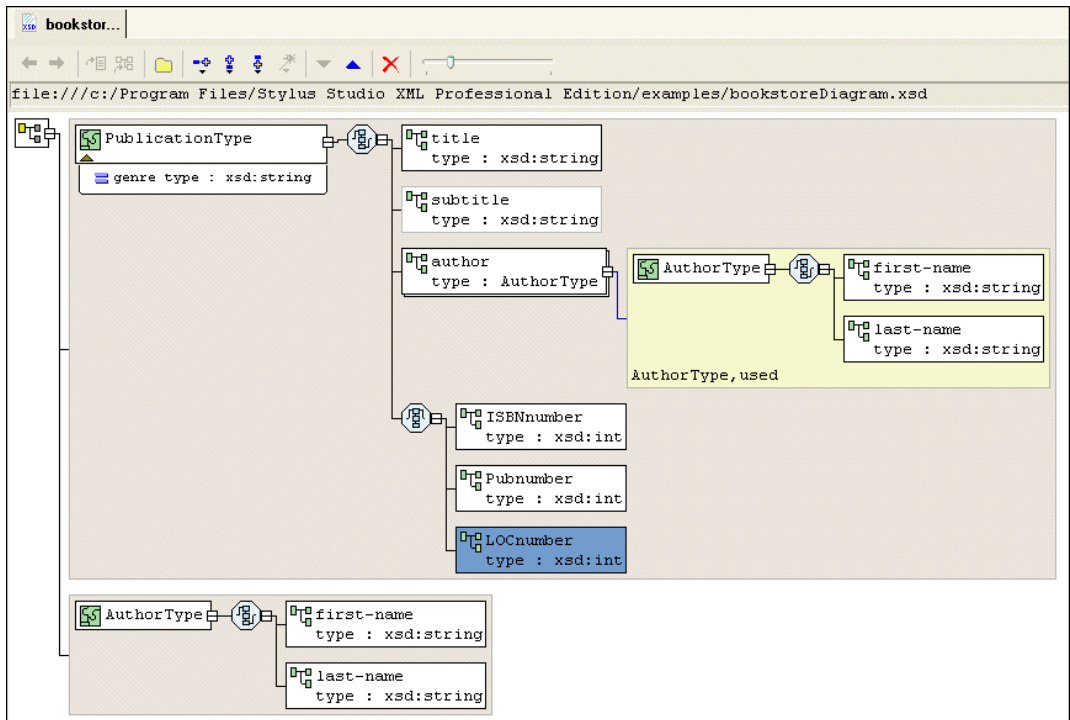



Figure 90. PublicationType complexType Fully Defined

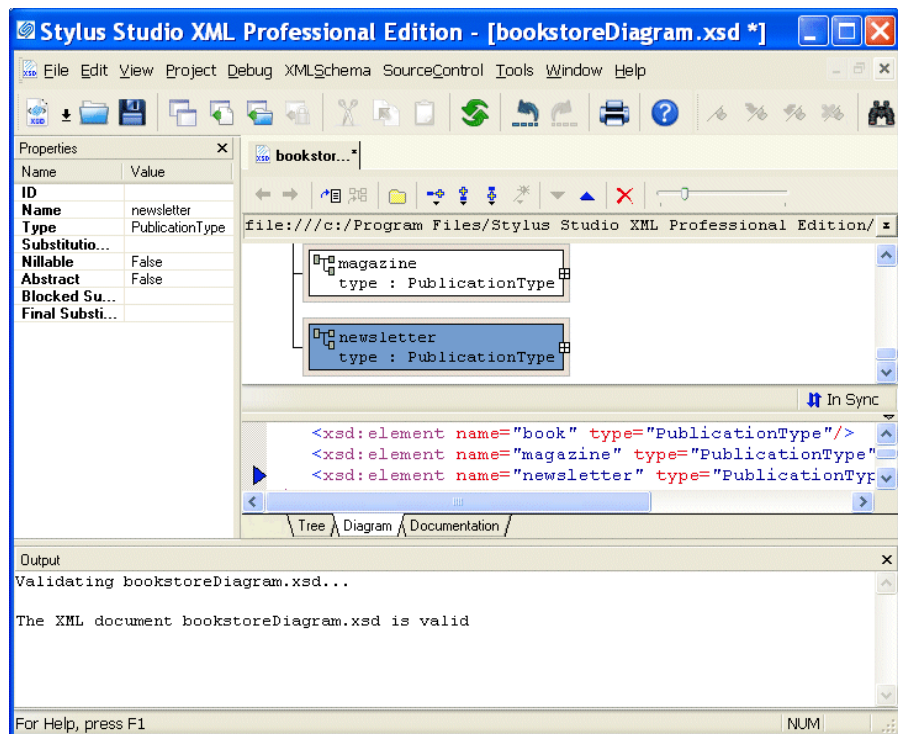
## Defining Elements of the Sample complexType in the Diagram View

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 91. 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 110
- “Using the File Explorer” on page 112
- “Dragging and Dropping Files in the Stylus Studio” on page 115
- “Other Ways to Open Files in Stylus Studio” on page 116
- “Adding File Types to Stylus Studio” on page 117

**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 912.

## 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	Convert to XML Editor
.dff	XML Diff Viewer
.dtd	DTD Schema Editor
.java	Java Debugger
.prj	Project framework
.rdbxml	DB-to-XML Data Source Editor
.wscc, .wsc	Web Service Call Composer
.xml	XML Editor
.xquery	XQuery Editor

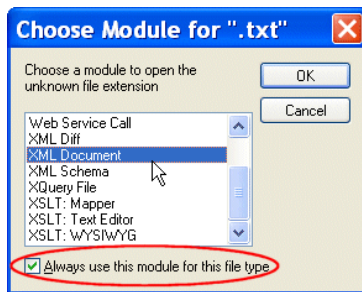
**Table 2. File Extensions and Associated Modules**

<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 117.

## 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 92. 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 117 for more information.

## Opening Files Stored on Berkeley DB XML

Stylus Studio provides access XML documents stored on Sleepycat Software’s Berkeley DB XML database. See [“Using Stylus Studio with Berkeley DB XML”](#) on page 817.

## 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).

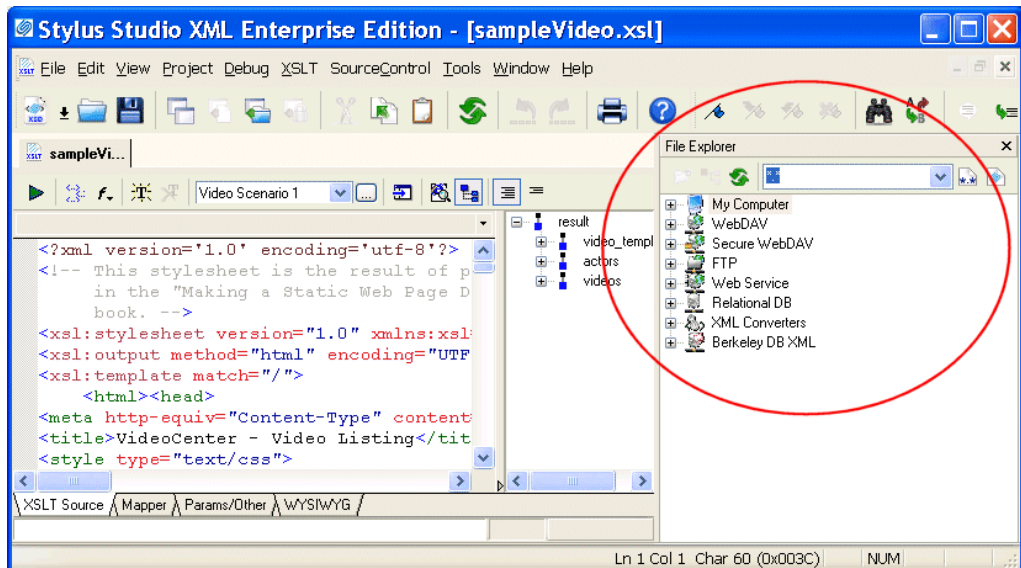


Figure 93. Stylus Studio File Explorer

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

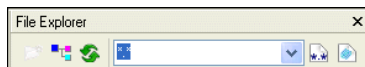
**Tip** **Open With** allows you to select the module you want to use to open the file.

- Drag and drop the file. See [Dragging and Dropping Files in the Stylus Studio](#) on page 115.

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 110 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 94. File Explorer Tool Bar**

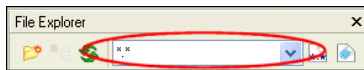
These features are summarized in the following table.

**Table 3. File Explorer Tools**

<i>Tool</i>	<i>Description</i>
 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: <pre>doc("file:///c:/Program Files/Stylus Studio XML Enterprise Edition 6/examples/simpleMappings/books.xml")/books/book/title</pre>
 Refresh	Refreshes the <b>File Explorer</b> 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: <code>.xml</code> , <code>.xsd</code> , <code>.dtd</code> , <code>.java</code> , <code>.conv</code> , and others.


## Working with the File Explorer Filter

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



**Figure 95. 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**

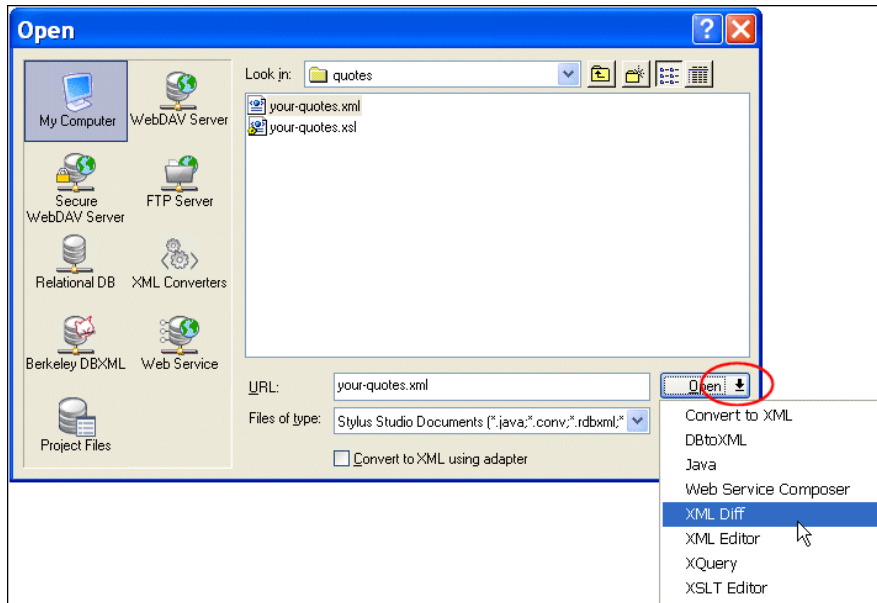
<i><b>If You Drop the File Here</b></i>	<i><b>Stylus Studio Does This</b></i>
On the document editor area (when no documents are open)	Opens the document editor associated with the type of file you selected. See <a href="#">Types of Files Recognized by Stylus Studio</a> on page 110. If the file type is not currently registered with Stylus Studio, you can register the file at this time using the <b>Choose Module for</b> dialog box. See <a href="#">Opening Unknown File Types</a> on page 111.
On the document editor tab area	Opens the document editor associated with the type of file you selected. See <a href="#">Types of Files Recognized by Stylus Studio</a> on page 110. If the file type is not currently registered with Stylus Studio, you can register the file at this time using the <b>Choose Module for</b> dialog box. See <a href="#">Opening Unknown File Types</a> on page 111.
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 <b>Project</b> 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 <a href="#">Opening Unknown File Types</a> on page 111.

## 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 110). 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 96](#).



**Figure 96. 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 111.

## 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.

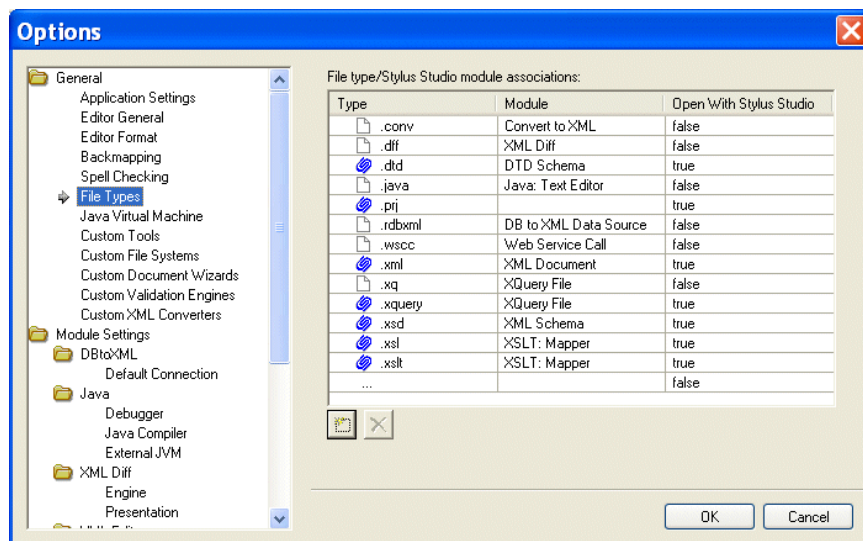




Figure 97. Associating File Types with Stylus Studio Editors

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.

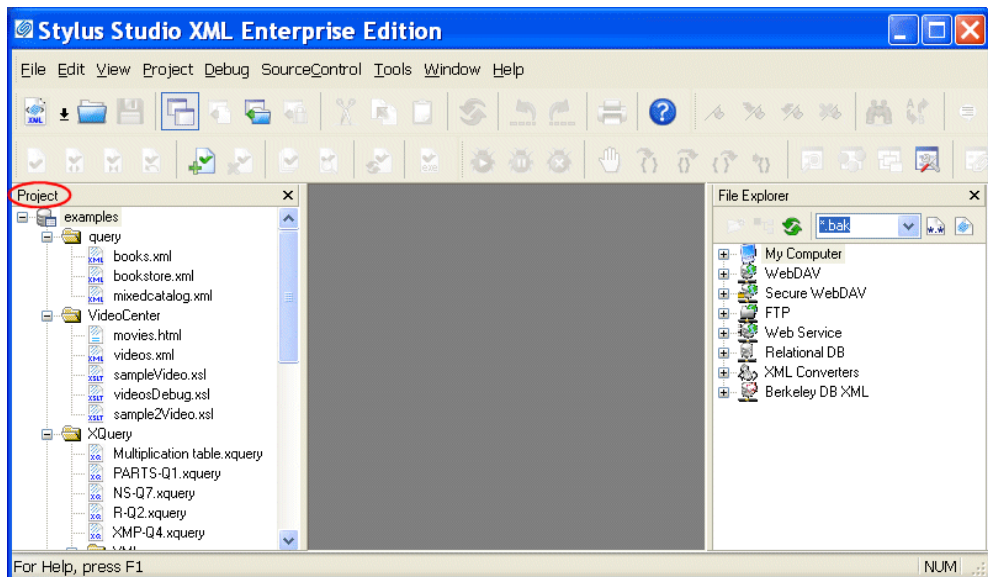
This section discusses the following topics:

- “[Displaying the Project Window](#)” on page 119
- “[Creating Projects and Subprojects](#)” on page 121
- “[Saving Projects](#)” on page 121

- “Opening Projects” on page 121
- “Adding Files to Projects” on page 122
- “Copying Projects” on page 123
- “Rearranging the Files in a Project” on page 124
- “Removing Files from Projects” on page 124
- “Closing and Deleting Projects” on page 124
- “Setting a Project Classpath” on page 125
- “Using Stylus Studio with Source Control Applications” on page 127

## 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 98. 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

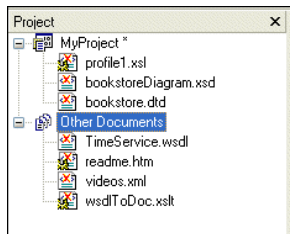
You can control whether the **Project** window displays absolute or relative path names for files in projects. The default display is relative names.

◆ **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 99. Other Documents Folder**

You can add these documents to a project at any time. See [“Adding Files to Projects”](#) on page 122.

---

## 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 122.

## 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.

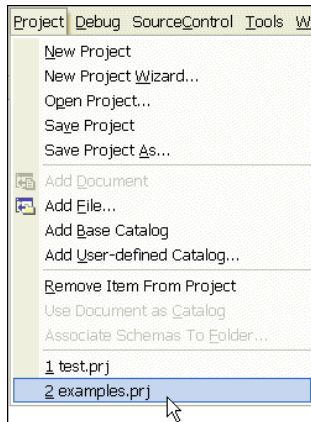
If the **Project** window is not visible when you open a project, Stylus Studio automatically displays the **Project** window.

◆ **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 100. 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 111 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.

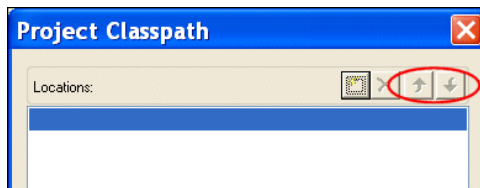
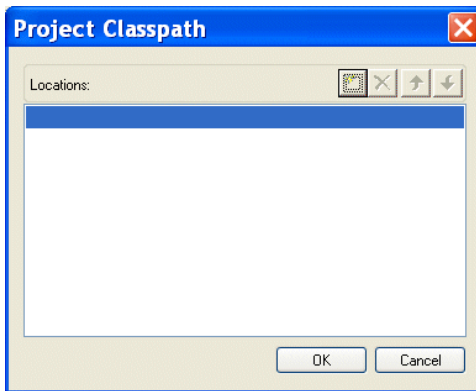


Figure 101. Use Arrow Buttons to Change the Order of Classpaths



## How to Set a Project Classpath

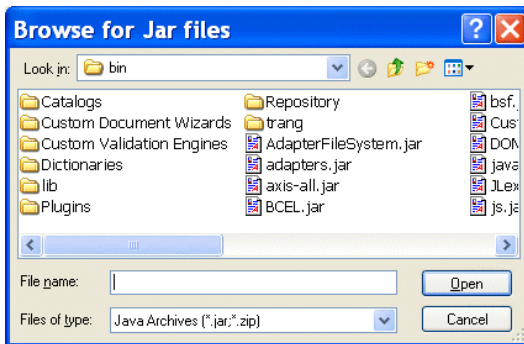
- ◆ **To set a classpath for a project:**
  1. Open the **Project Window** if it is not already displayed.
  2. Right-click the project node.  
The project shortcut menu appears.  
*Alternative:* Select **Project > Set Classpath** from the Stylus Studio menu.
  3. Select **Set Project Classpath**.

The **Project Classpath** dialog box appears.



**Figure 102. Project Classpath Dialog Box**

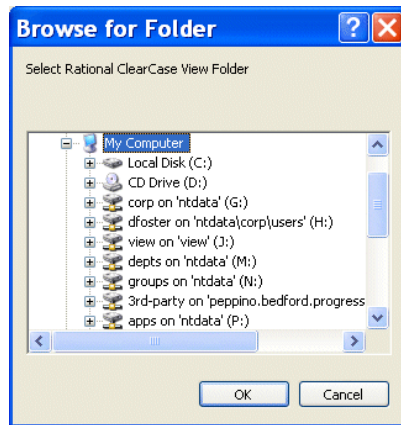
4. Click the browse folders () button.  
A new entry field appears in the **Locations** list box. Two buttons appear to the right of the entry field.
5. To add a JAR file to the classpath, click the browse jar files button ().  
Stylus Studio displays the **Browse for Jar Files** dialog box.



**Figure 103. 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 104. 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 128
- [“Prerequisites”](#) on page 128
- [“Using Stylus Studio with Microsoft Visual SourceSafe”](#) on page 129
- [“Using Stylus Studio with ClearCase”](#) on page 131
- [“Using Stylus Studio with Zeus CVS”](#) on page 134
- [“Specifying Advanced Source Control Properties”](#) on page 135

### 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, xslt, 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.

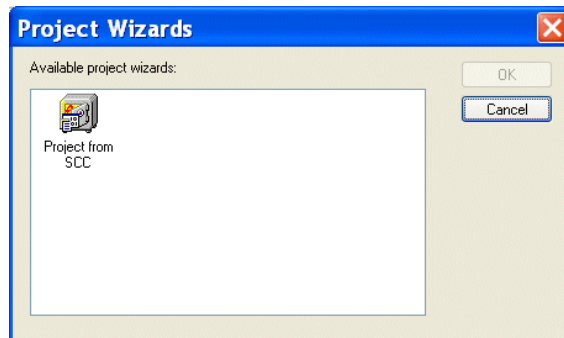
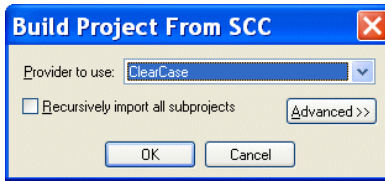


Figure 105. Project Wizards Dialog Box

2. Click **Project from SCC**, and click the **OK** button.

The **Build Project From SCC** dialog box appears.



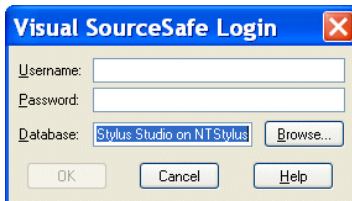
**Figure 106. Build Project From SCC Dialog Box**

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 128 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 135.

5. Click the **OK** button.

The **Visual SourceSafe Login** dialog box appears:

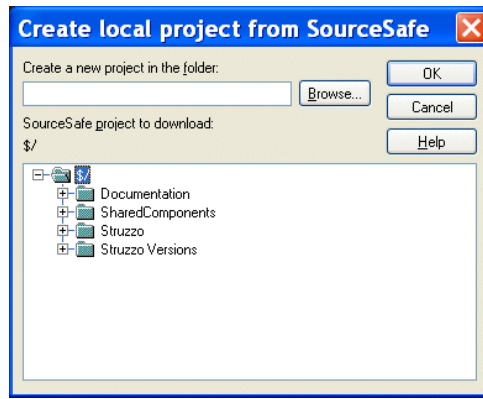


**Figure 107. Visual SourceSafe Login Dialog Box**

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 108. 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.

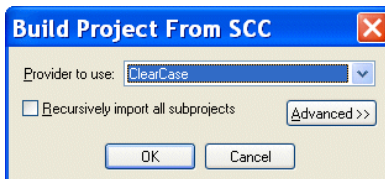
2. From the Stylus Studio menu bar, select **Project > New Project Wizard**.

The **Project Wizards** dialog box appears.



**Figure 109. Project Wizards Dialog Box**

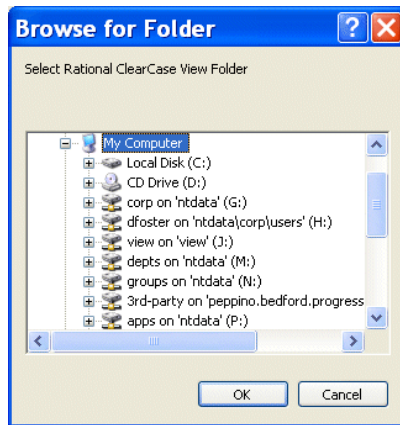
3. Click **Project from SCC**, and click the **OK** button.  
The **Build Project From SCC** dialog box appears.



**Figure 110. 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 128 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 135.
6. Click the **OK** button.

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




**Figure 111. 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 *Project.n*. 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.

- Click the **OK** button.

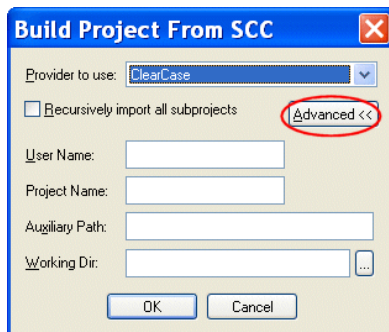
Stylus Studio downloads the selected files and places them in the directory you specified in the **Working Dir** field. If you move these files from this directory, you must specify the new directory that contains the files in the **Local Project Path** field of the **Source Control Properties** dialog 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 112. 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 136
- “[Showing/Hiding Tool Bar Groups](#)” on page 137
- “[Changing Tool Bar Appearance](#)” on page 138

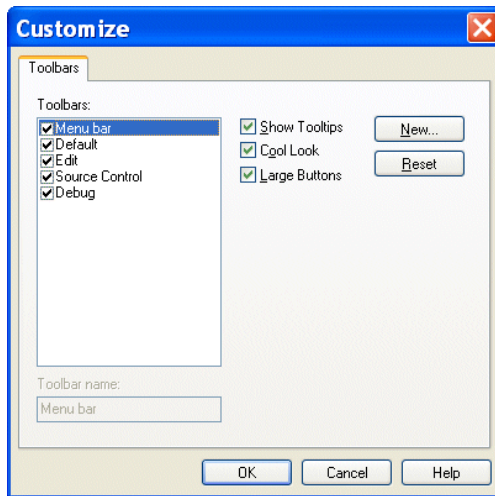
### Tool Bar Groups

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 113. 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 as buttons (the **Cool Look**) or flat panels
- 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 **Cool Look** to toggle the display of the tool bar between one that is rendered as buttons and one that is rendered as flat panels.
4. Click **Large Buttons** to toggle the size of the tool bar buttons.
5. Optionally, click the **Reset** button to restore default settings.
6. Click the **OK** button.

## Specifying Stylus Studio Options

Stylus Studio allows you to set a variety of options that apply to your application, your Java implementation, and the Stylus Studio modules themselves. This section covers the following topics:

- [“Modifying Java Options”](#) on page 139
- [“Setting Module Options”](#) on page 142
- [“Defining Custom Tools”](#) on page 144



## Modifying Java Options

Stylus Studio allows you to modify settings for

- Java virtual machine (JVM) used by Stylus Studio
- Stylus Studio Java debugger (see [“Debugging Java Files”](#) on page 489 for more information on this topic)
- Java compiler
- External Java virtual machine

If you do not make any changes to these settings, Stylus Studio looks in the registry to determine the locations of your Java components.


This section covers the following topics:

- [“About Java Virtual Machine Options”](#) on page 139
- [“About Java Compiler Options”](#) on page 140
- [“About External JVM Options”](#) on page 141
- [“How to Modify Java Settings”](#) on page 142

### About Java Virtual Machine Options

When Stylus Studio needs to execute Java code as part of the application of a stylesheet, it loads the Java Virtual Machine (JVM) that is in the run-time library specified in the **Java Virtual Machine** options page. To open this page, select **Tools > Options** from the Stylus Studio menu bar, and then click **Java Virtual Machine** (under **General**).

On the **Java Virtual Machine** page, you can modify the following options:

- **Enabled** indicates whether Stylus Studio loads a JVM. If you do not need a JVM, you can disable this option so that Stylus Studio requires less memory.
- **Runtime Library** location. To specify an alternate location, type a new path or click **Browse** .
- **JDK Home Directory** for your Java Development Kit (JDK). To specify a new home directory, type the value or click **Browse**.

- **Classpath** used by JVM in Stylus Studio. To add a directory or .jar file to this classpath, click **Browse**. For Stylus Studio to be able to find a Java class, its location must be specified in this classpath.

**Note**

This value does not affect the CLASSPATH environment variable. The default value, `$(classpath)`, specifies that you want to use the contents of the CLASSPATH environment variable. There is no requirement for this classpath and the classpath used by the Stylus Studio Java compiler to be identical. However, it is good practice for them to be the same.

- **Parameters** specifies the parameters for starting the JVM in the context of Stylus Studio.

**Note**

As a rule, you should never change the default value. This option exists to accommodate unusual configurations. In such situations, Stylus Studio Technical Support might instruct you to change this value.

### Resetting Options

If you modify the Stylus Studio **Java Virtual Machine** options and then decide you want to return to the default values, click **Auto detect** to reset the options.

### About Java Compiler Options

When you use Stylus Studio to compile Java classes, you can specify a number of options on the **Java Compiler** page. To open this page, select **Tools > Options** from the Stylus Studio menu bar, and then click **Module Settings > Java > Java Compiler**.

On the **Java Compiler** page, you can modify the following options:

- **Java Compiler** is the location of the Java compiler that Stylus Studio uses. To change the location, type a path in the field or click **Browse**  to the right of the field.
- **ClassPath** used by Java compiler in Stylus Studio. To add a directory or .jar file to this classpath, click **Browse** to the right of the field. For Stylus Studio to be able to find a Java class, its location must be specified in this classpath.

**Note**

This value does not affect the CLASSPATH environment variable. The default value, `$(classpath)`, specifies that you want to use the contents of the CLASSPATH environment variable. There is no requirement for this classpath and the classpath used by the Stylus Studio JVM to be identical. However, it is good practice for them to be the same.

- **Parameters** specifies the parameters for running the Java compiler. Typically, you want to specify the `-g` parameter, which instructs the compiler to add debugging information to the generated `.class` file.

### About External JVM Options

When Stylus Studio executes Java code as a standalone application, it uses the JVM that is specified in the **External JVM** options page. To open this page, select **Tools > Options** from the Stylus Studio menu bar, and then click **Module Settings > Java > External JVM**.

On the **External JVM** page, you can modify the following options:

- **External JVM** identifies the location of the JVM you want to run.
- **Additional Classpath** used by this external JVM. To add a directory or `.jar` file to this classpath, click **Browse**  to the right of the field. For Stylus Studio to be able to find a Java class, its location must be specified in this classpath.

**Note** This value does not affect the `CLASSPATH` environment variable. The default value, `$(classpath)`, specifies that you want to use the contents of the `CLASSPATH` environment variable. There is no requirement for this classpath and the classpath used by the Stylus Studio Java compiler to be identical. However, it is good practice for them to be the same.

- **Parameters** specifies the parameters for starting this external JVM.

**Note** As a rule, you should never change the default value. This option exists to accommodate unusual configurations. In such situations, Stylus Studio Technical Support might instruct you to change this value.

- **Display Console Window** allows you to view processing information in a console window.

### Resetting Options

If you modify the Stylus Studio **External JVM** options and then decide you want to return to the default values, click **Auto detect** to reset the options.

### How to Modify Java Settings

◆ **To modify Java settings:**

1. From the Stylus Studio menu bar, select **Tools > Options**.
2. Click one of the following:
  - **General > Java Virtual Machine**
  - **Module Settings > Java > Debugger** — See [“Debugging Java Files”](#) on page 489.
  - **Module Settings > Java > Java Compiler**
  - **Module Settings > Java > External JVM**
3. Make your changes and click **OK**.

If the JVM is not already loaded, any changes you make take effect immediately. If the JVM is already loaded in Stylus Studio, you must restart Stylus Studio for the changes to take effect.

### 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 195 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 795 for more information.

## XSLT Editor

Module settings for the XSLT Editor let you specify external XSLT processors, settings used by the **Mapper** and **WYSIWYG** tabs, 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 additional path for Xalan-J processor
- Default additional classpath for Xalan-J processor
- Additional JVM options for Xalan-J processor
- 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 454 for more information on using the XSLT Mapper.

Click **WYSIWYG** to specify settings Stylus Studio uses to define tab stops and new lines in the HTML it generates. See [“Creating Stylesheets That Generate HTML”](#) on page 369 for more information on using the XSLT WYSIWYG editor.

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 “[Modifying Java Options](#)” on page 139.

## 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.

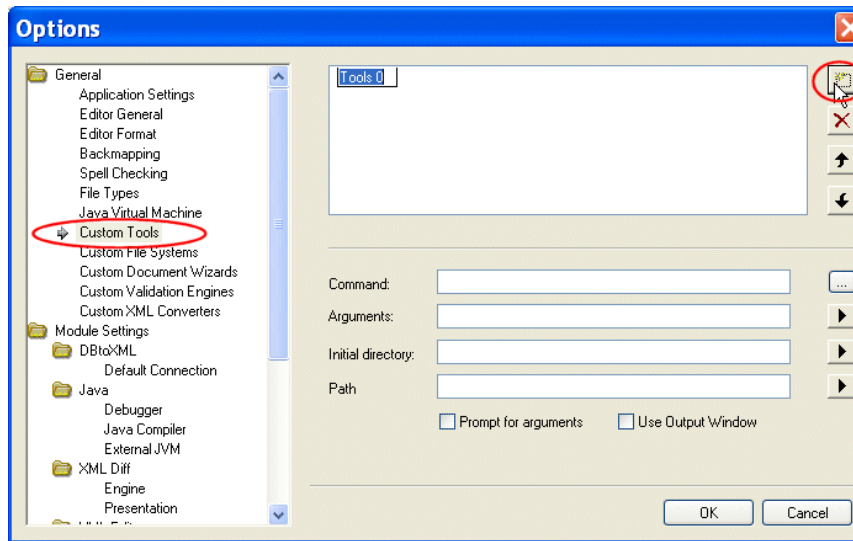



Figure 114. Defining a Custom Tool

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.

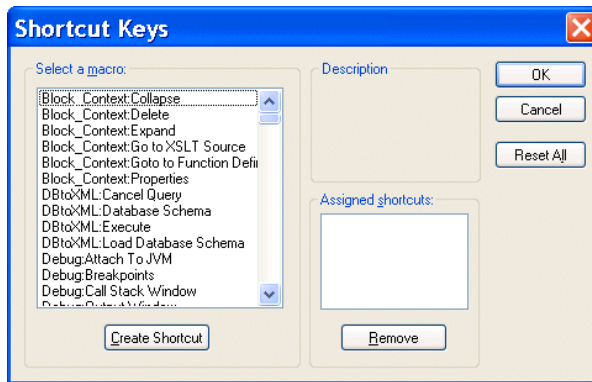
## Defining Keyboard Shortcuts

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 115. Defining Shortcut Keys**

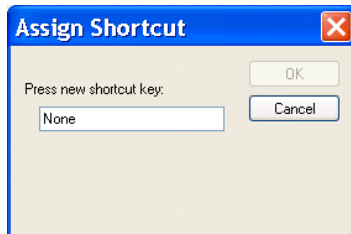
2. In the **Select a macro:** field, select the macro for which you want to define a shortcut.

**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 116. 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.



- Click **OK** in the **Shortcut Keys** dialog box.

## Deleting a Keyboard Shortcut

### ◆ To delete a shortcut:

- From the Stylus Studio menu bar, select **Tools > Keyboard**. The **Shortcut Keys** dialog box appears.
- In the **Select a macro:** field, select the macro you want to delete a shortcut for.
- In the **Assigned shortcuts** field, click the shortcut you want to remove.
- Click **Remove**.
- 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	<a href="#">“Invoking Stylus Studio from the Command Line”</a> on page 148
StylusDiff	Differs two XML documents	<a href="#">“Running the Diff Tool from the Command Line”</a> on page 223
StylusValidator	Validate XML	<a href="#">“Validating XML from the Command Line”</a> on page 150
StylusXq1	Execute an XQuery	<a href="#">“Executing an XQuery from the Command Line”</a> on page 149
StylusXslt	Apply a stylesheet	<a href="#">“Applying a Stylesheet from the Command Line”</a> on page 148

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

## 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**

<i>Parameter</i>	<i>Description</i>
<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</i> or <i>XQuery</i> parameter.
[ <i>stylesheet</i> or <i>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**

<b>Parameter</b>	<b>Description</b>
<code>[-out &lt;output file&gt;]</code>	File to which the XSLT result will be written. The default is stdout.
<code>[-param name=value]</code>	The name-value pair of a parameter in the stylesheet specified in the <code>stylesheet</code> parameter.
<code>[-fop]</code>	Invokes the Apache Formatting Objects Processor (FOP) to post-process the XSLT result.
<code>[-print]</code>	Sends the result of the transformation to the default printer.
<code>[-in &lt;input file&gt;]</code>	The path of the XML document to which you want to apply the stylesheet specified in the <code>XSLT stylesheet</code> parameter.
<code>&lt;XSLT stylesheet&gt;</code>	The path of the XSLT you want to apply to the document specified in the <code>-in</code> parameter.

## Executing an XQuery from the Command Line



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

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**

<i>Parameter</i>	<i>Description</i>
-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 XQuery file parameter.
-i	Indents the XQuery result.
-debug host[:port]	Debugs the query using the debug server specified by the host and, optionally, port parameters.
XQuery file	The path of the XQuery you want to execute against the file specified in the -in <source> 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 StylusXq1 command.

**Table 11. StylusValidator Command Line Parameters**

<i>Parameter</i>	<i>Description</i>
[-q]	Quiet mode – errors are not printed to stdout.
[-noval]	Checks only for well-formedness. Does not check for errors.

**Table 11. StylusValidator Command Line Parameters**

<i>Parameter</i>	<i>Description</i>
<code>[-schema <i>file</i>]</code>	Validates the XML document against the XML Schema specified in the <i>file</i> parameter.
<i>filename</i>	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	<a href="#">Changing the Schema Refresh Interval</a> on page 152 <a href="#">Checking for Modified Files</a> on page 153

**Table 12. Performance Symptoms**

<b>Symptom</b>	<b>See</b>
Errors or crashes during XSLT processing	<a href="#">Changing the Recursion Level or Allocated Stack Size</a> on page 153
Stylus Studio is slow to start	<a href="#">Automatically Opening the Last Open Files</a> on page 154

## 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 941.

**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 912.  
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 954.
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 912.
4. Click **OK**.



## Chapter 2    **Editing and Querying XML**

Stylus Studio makes it easy to 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. At any time, you can view the schema that is associated with your XML document.

You can compose XQueries using the Stylus Studio XQuery editor, which allows you to write XQueries in text or compose them graphically using Stylus Studio's mapper technology, or graphically, using the XQuery mapper. See [“Working with XQuery in Stylus Studio”](#) on page 695.

This section covers the following topics:

- [“Creating XML Documents”](#) on page 156
- [“Converting Text Files to XML Documents”](#) on page 156
- [“Converting HTML to XML Documents”](#) on page 163
- [“Updating XML Documents”](#) on page 163
- [“Updating XML Documents Using the Text Editor”](#) on page 166
- [“Updating DOM Tree Structures”](#) on page 176
- [“Updating XML Documents Using the Grid Tab”](#) on page 179
- [“Diffing Folders and XML Documents”](#) on page 195
- [“Using Schemas with XML Documents”](#) on page 226
- [“Querying XML Documents Using XPath”](#) on page 229
- [“Moving Around in XML Documents”](#) on page 231
- [“Printing XML Documents”](#) on page 232
- [“Saving XML Documents”](#) on page 233

# Creating XML Documents

To create an XML document, from the Stylus Studio menu bar, select **File > New > XML Document**.

Stylus Studio displays a new window that contains an empty XML document. The document contains only the XML declaration:

```
<?xml version="1.0"?>
```

There are several ways to update your XML document. See [“Updating XML Documents”](#) on page 163.

## Other Ways to Create XML

You can also create XML using

- Document wizards, to convert text and HTML, for example, to XML. There are also converters that allow you to create XML from a DTD, from an XML Schema, and from ADO. See [“Converting Text Files to XML Documents”](#) on page 156 and [“Converting HTML to XML Documents”](#) on page 163, later in this section.
- User-defined converters, built using Stylus Studio’s Convert to XML module. See [“Converting Non-XML Files to XML”](#) on page 235.
- Built-in converters, like converters for EDI. See [“Using Adapters in Stylus Studio”](#) on page 280.

# Converting Text Files to XML Documents

Stylus Studio provides document wizards that convert Comma Separated Value (CSV) and fixed-width text files to XML documents. A CSV or fixed-width file is an ASCII text file that can be treated as a table. That is, it implements the concepts of rows and columns.

This section discusses the following topics:

- [“Alternative to Document Wizards”](#) on page 157
- [“About CSV File Contents”](#) on page 157
- [“Structure of Resulting XML Documents”](#) on page 158
- [“About the Default Values”](#) on page 159
- [“Running the Convert CSV to XML and Convert Fixed-Width to XML Document Wizards”](#) on page 159

## Alternative to Document Wizards

Stylus Studio's Convert to XML is a module you use to build converters that convert non-XML files to XML documents. Convert to XML converters can handle a number of file formats, including text, binary, and EDI. Another advantage of using Convert to XML to build your own converters is that the converter can be used to open a non-XML file as an XML document anywhere in Stylus Studio – as the source document for XQuery Mapper, for example. Once you build the converter, the conversion process takes place automatically any time you open a file with it.

See [“Converting Non-XML Files to XML”](#) on page 235 to learn more about the Convert to XML module.

## About CSV File Contents

In a CSV file, a new line character flags each row of the table. A comma delimits each column within that row. For this scheme to work, all lines in the file must contain the same number of values. For example:

```
Albert,Archer,1 Alpine St,Acton,MA
Blaine,Baker,2 Bedford Rd,Burlington,MA
Claire,Claus,3 Charlotte Ave,Cambridge,MA
```

In a fixed-width file, a new line character also flags each row of the table. However, the width of a particular column is the same in each row. For example:

```
AlbertArcher1 Alpine St Acton MA
BlaineBaker 2 Bedford Rd BurlingtonMA
ClaireClaus 3 Charlotte AveCambridge MA
```

## Delimiting Commas

If a value in a CSV file contains a comma, you must enclose the value in a pair of quotation characters. You can choose the character that you want to use as the quotation character. The default is to use double quotes:

```
Dan,Davidson,"4 Denver St, Apt 4",Dover,MA
```

If the character you use as the quotation character appears in a value that is enclosed in quotation characters, you must double each enclosed quotation character. For example:

```
Eugene,"""Ed"" Everett, Jr.",5 Easy St,Edgartown,MA
```

If you want, you can include the names of the columns in the file. If you do, they must be the first row of data. For example:

```
First,Last,Street,City,State
Albert,Archer,1 Alpine St,Acton,MA
Blaine,Baker,2 Bedford Rd,Burlington,MA
Claire,Claus,3 Charlotte Ave,Cambridge,MA
```

If there are any invalid characters in XML tag names, the wizard replaces the invalid character with an underscore.

## Structure of Resulting XML Documents

By default, converting a text file results in an XML document with the following format:

```
<document>
  <row>
    <value1>value</value1>
    <value2>value</value2>
    <value3>value</value3>
    ...
  </row>
  <row>
    <value1>value</value1>
    <value2>value</value2>
    <value3>value</value3>
    ...
  </row>
  <row>
    <value1>value</value1>
    <value2>value</value2>
    <value3>value</value3>
    ...
  </row>
  . . .
</document>
```

The document wizard replaces *value* with the actual value that is in the imported file. If you want, you can specify that you want the document wizard to map the values to attributes of the row elements. The output would look like this:

```
<document>
  <row value1="value" value2="value" value3="value"></row>
  <row value1="value" value2="value" value3="value"></row>
  <row value1="value" value2="value" value3="value"></row>
  . . .
</document>
```

## About the Default Values

Stylus Studio uses these default values for converted files:


- document is the root element node
- row is the row element name
- Column names are generated as value1, value2, value3, and so on
- Column values are mapped to children of row elements
- Comma (,) is the delimiter character
- Double quote (") is the quote character
- UTF-8 is the default encoding of the file being imported

When you convert a file, you can change any of these values. At any time, you can select **Make Default** to store the current settings in the Windows registry. Stylus Studio uses these new default values as the initial settings the next time the **Convert** dialog box is opened.

If you want to reset the default values to the values that were in place when you installed Stylus Studio, click **Reset Default** in the **Convert CSV to XML** or **Convert Fixed-Width to XML** dialog box.

## Running the Convert CSV to XML and Convert Fixed-Width to XML Document Wizards

### ◆ To run the **Convert CSV to XML and Fixed-Width to XML document wizard**:

1. In the Stylus Studio menu bar, select **File > Document Wizards**. The **Document Wizards** dialog box appears.
2. In the **Document Wizards** dialog box, in the **XML Editor** tab, double-click one of the following according to the format of the file being converted:
  - **CSV to XML**. The **Convert CSV to XML** dialog box appears.
  - **Fixed-Width to XML**. The **Convert Fixed-Width to XML** dialog box appears.
3. In the **Input URL** field, type the name of the text file you want to convert, or click **Browse**  to navigate to and select the file you want to convert.
4. In the **Input Encoding** field, select the character encoding of the text file being converted. The default for your platform is typically UTF-8 or UTF-16. If you do not specify a value, the wizard uses the default for your platform.

5. In the **Root element name** field, type the name you want for the root element in the XML document. The default is `document`.
6. In the **Row element name** field, type the name you want for the row elements in the XML document. The default is `row`.
7. In the **Column names are** group box, click one of the following:
  - **Generated** indicates that you want Stylus Studio to generate column names. The default column names are `value1`, `value2`, `value3`, and so on.
  - **In file** indicates that the first line in the text file contains the column names.
  - **User defined** indicates that you are entering the names of the columns in the text area. Use the buttons to the right of the text area to ensure that the column names are in the correct order.

For more information on specifying user defined columns, see [“Specifying User-Defined Columns”](#) on page 161.


8. In the **Map value to** group box, click **Child element** to map the column values to child elements of the row elements. For example:

```
<document>
  <row>
    <value1>First Value</value1>
    <value2>Second Value</value2>
  </row>
</document>
```

Click **Attribute** to map the column values to attributes of the row elements. For example:

```
<document>
  <row value1="First Value" value2="Second Value"></row>
</document>
```

9. If you are converting a fixed-width file, go to [Step 12](#). You do not need to specify delimiting or quote characters for fixed-width files.
10. In the **Delimiter** group box, accept the default delimiter character, which is a comma (,), select another common delimiter, or specify some other delimiter. The delimiter is the character that appears between values in the text file. Stylus Studio automatically displays the hexadecimal value for the character you specify. If you prefer, you can enter the hexadecimal value for the character you want. Stylus Studio automatically displays the correct character in the character field.

11. In the **Quote** group box, accept the default quotation character, which is a double quote ("), select another common quotation character, or specify that you want to use some other character. Stylus Studio automatically displays the hexadecimal value for the character you specify. If you prefer, you can enter the hexadecimal value for the character you want. Stylus Studio automatically displays the correct character in the character field.
12. To specify that you want the current settings to be the default settings, click **Make Default** so that it is selected. Stylus Studio copies the current settings to the registry.
13. To specify that you want the default settings to be the values that were in place when you installed Stylus Studio, click **Reset Default** so that it is selected. Stylus Studio changes the default values to the values specified in [“About the Default Values”](#) on page 159.
14. Click **OK**. Stylus Studio converts the source document to XML and displays the resulting XML document in the XML editor.
15. In the Stylus Studio tool bar, click **Save** , to give the new XML document a name.

## Specifying User-Defined Columns

The procedure for specifying user defined columns varies slightly based on whether you are converting a CSV or a fixed-width file to XML.

### Column Names for CSV Files

When you specify a user defined column for a CSV file, you need to supply a name for each column, as shown here.

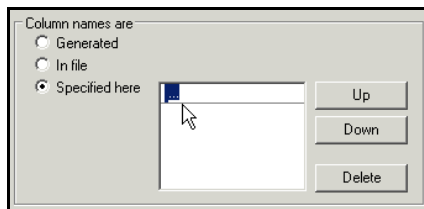


Figure 117. Specifying Column Names in CSV Files

- ◆ **To specify a column name for a CSV file:**
  1. Click the entry field next to **User defined**.
  2. Type the name for the first column.

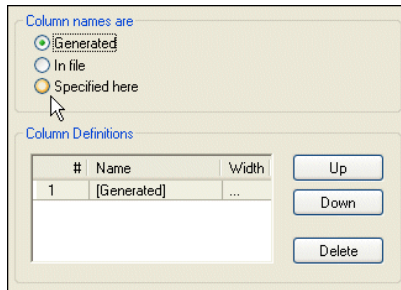
3. Press Enter.
4. Repeat [Step 2](#) and [Step 3](#) for each additional column.

**Tip**

You can change column order using the **Up** and **Down** buttons.

### Column Names for Fixed-Width Files

When you specify a user defined column for a fixed-width file, you need to specify both the name and the width of the column, as shown here:



**Figure 118. Specifying Column Names in Fixed-Width Files**

◆ **To specify a column name for a fixed-width file:**

1. Click the **Specified here** button.
2. Click the **Name** field in the **Column Definitions** group box.
3. Type a name in the field.
4. Press Tab.  
The name value is entered, and a new row for the next column is added to the **Column Definitions** group box. The cursor appears in the **Width** field.
5. Enter a value for the column width and press Tab or Enter.  
The column definition for the first column is specified in the wizard.
6. Repeat [Step 2](#) through [Step 5](#) for any additional columns you need to define.



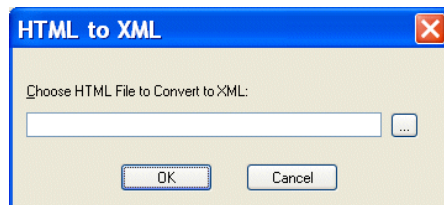
## Converting HTML to XML Documents

You can create an XML document from an HTML file using the HTML to XML document wizard.

**Tip** Stylus Studio also has a document wizard that converts HTML to XSLT. See [Creating a Stylesheet from HTML](#) on page 354.

◆ **To run the HTML to XML document wizard:**

1. Select **File > Document Wizards** from the menu.  
The **Document Wizards** dialog box appears.
2. Double-click **HTML to XML** (or select the HTML to XML icon and click **OK**).  
The **HTML to XML** dialog box appears.



**Figure 119. HTML to XML Dialog Box**

3. Enter the name of the HTML file you want to convert to XML in the **Choose HTML File to Convert to XML** field.
4. Click **OK**.  
Stylus Studio opens the converted HTML file as an untitled XML document in the XML Editor.

## Updating XML Documents



The XML editor **Grid** tab is available only in Stylus Studio XML Enterprise Edition and Stylus Studio XML Professional Edition.

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 164
- [“Saving Your Work”](#) on page 164
- [“Ensuring Well-Formedness”](#) on page 165
- [“Reverting to Saved Version”](#) on page 165
- [“Updating Java Server Pages as XML Documents”](#) on page 165

### 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 229.

### For More Information

To learn more about a specific XML view, see one of the following sections:

- [“Updating XML Documents Using the Text Editor”](#) on page 166
- [“Updating DOM Tree Structures”](#) on page 176
- [“Updating XML Documents Using the Grid Tab”](#) on page 179

### 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**.

## Updating XML Documents 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.

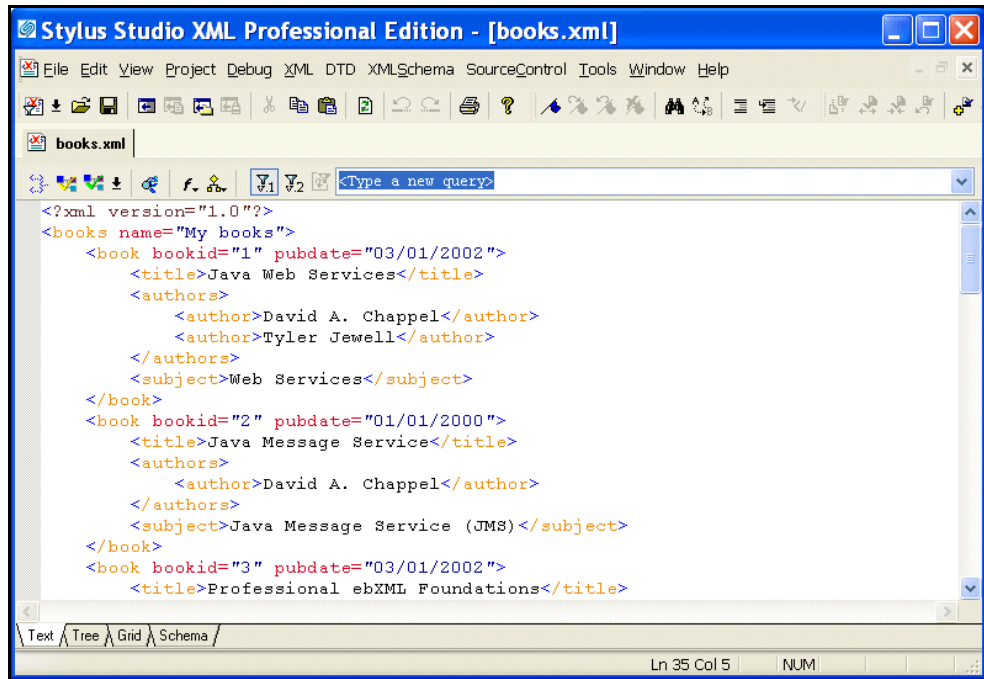


Figure 120. Text Tab in the XML Editor

This section covers the following topics:

- [Common Text Editing Functions and Tools](#) on page 166
- [Use of Colors in the Text Tab](#) on page 169
- [Using the Spell Checker](#) on page 171

### Common Text Editing Functions and Tools

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.

Following are some of Stylus Studio's main editing features.

### Sense:X


Sense:X prompts you with the possible tags that you can insert at a given location. The Sense:X feature is available if the document is associated with a schema. As soon as you type a tag open bracket, Stylus Studio displays a scrollable list of the elements that are allowed at that location. Double-click the tag you want. For example, suppose you have a 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

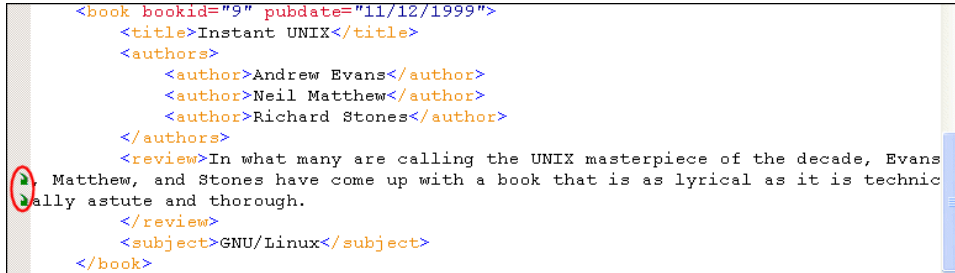
Indent XML tags to show the hierarchy relationships. Click **Indent XML Tags** . Stylus Studio indents all text in the active XML document window.

**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 also turn on line wrap manually, 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 121](#), identify lines that have wrapped.

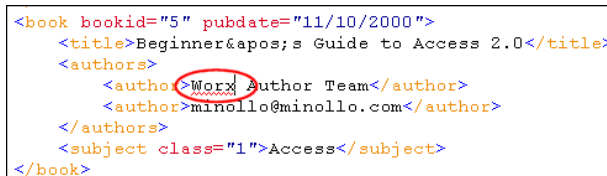
A screenshot of the Stylus Studio XML editor. The XML code is displayed in a monospaced font. The code includes a root element <book> with attributes bookid="9" and pubdate="11/12/1999". It contains sub-elements for <title>, <authors>, <review>, and <subject>. The <review> text is wrapped across two lines. Two green arrows on the left margin point to the start of the wrapped lines. The XML code is as follows:

```
<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 121. 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 122](#).


A screenshot of the Stylus Studio XML editor. The XML code is displayed in a monospaced font. The code includes a root element <book> with attributes bookid="5" and pubdate="11/10/2000". It contains sub-elements for <title>, <authors>, and <subject>. The word "Worx" in the <author> tag is underlined with a red squiggly line. The XML code is as follows:

```
<book bookid="5" pubdate="11/10/2000">
  <title>Beginner'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 122. Typographical Errors Are Highlighted by the Spell Checker**

For more information, see [Using the Spell Checker](#) on page 171.

## 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

Enclose selected text in comment tags. 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 comment element and its contents, and click **Comment/Uncomment Selection**.

## 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 484.

## Search

Search for and replace text you specify. Click **Find**  or **Replace**  in the tool bar. You can also enable Find by pressing Ctrl + F.

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 pressing the down arrow when the **Find what** field is active.

## 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>
Royal blue	Markup
Black	XML declaration and text node contents
Pale blue	Schema definition
Purple	Element names defined in the DTD
Red	Attribute names

**Table 13. Text Colors in Stylus Studio**

<i>Color</i>	<i>Type of Data</i>
Dark blue	Attribute values
Orange	Element names not defined in the DTD

### How to Change Text Colors

You can set colors differently for the text editors associated with different document types (XML, XQuery, XSLT, and so on).

◆ **To change text 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 elements as appropriate.
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.

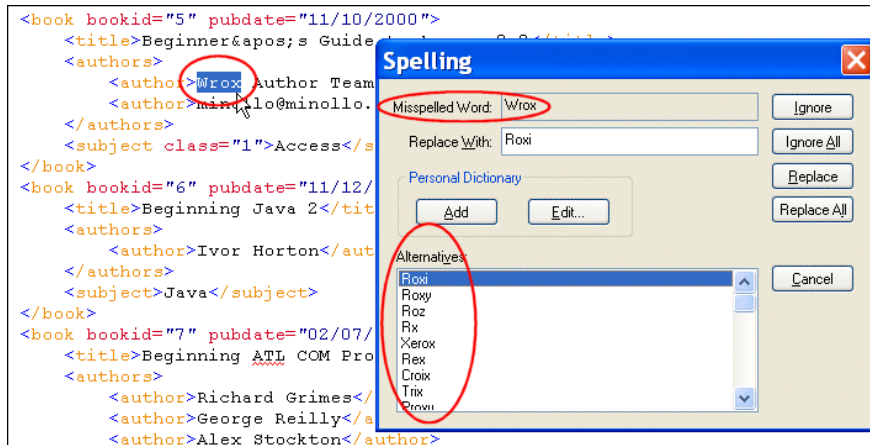


Figure 123. Stylus Studio’s Spell Checker

### 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 122.

**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

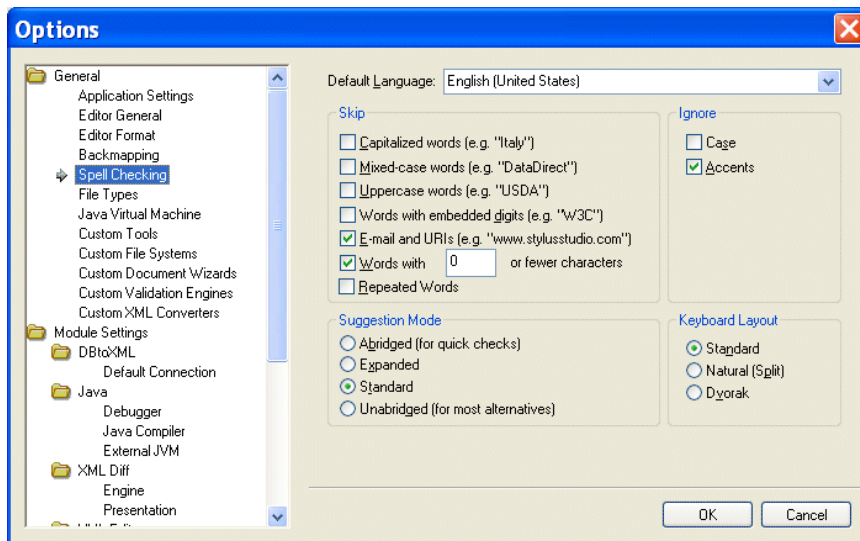
At any time, you can manually spell check a document by selecting **Tools > Check Spelling** from the Stylus Studio menu. When you do this, Stylus Studio starts the Spell Checker, which reads through the current document. When it finds a possible typographical error, Stylus Studio displays the **Spelling** dialog box, as shown in Figure 123.

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 124. 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 <b>Replace</b> . Click <b>Replace All</b> to replace all occurrences of that word.  You can also replace the misspelled word with another word selected from the <b>Alternatives</b> list box, or with a word you type in the <b>Replace with</b> field.
Ignore the misspelled word	Click <b>Ignore</b> . Click <b>Ignore All</b> to ignore all occurrences of that word.
Add the misspelled word to the personal dictionary	Click <b>Add</b> .
Edit the personal dictionary	Click <b>Edit</b> . See <a href="#">Using the Personal Dictionary</a> on page 174 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.

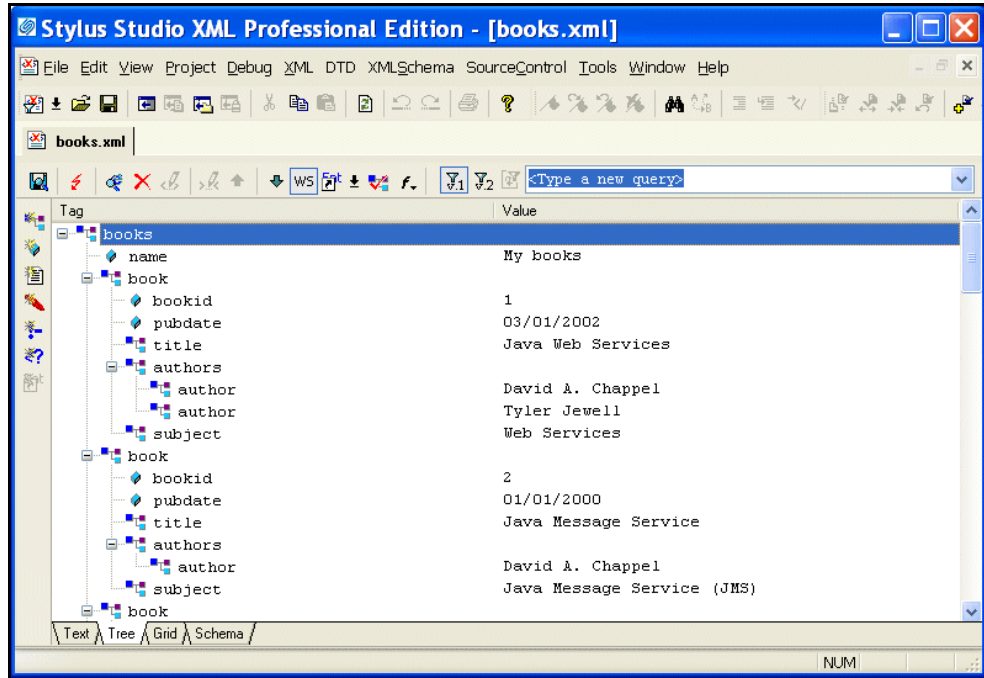
5. Click **Save**.

The contents of the personal dictionary is saved to the text file.


6. Click the **Close** button.

## 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 125. 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 177
- [“Adding a Node in the Tree View”](#) on page 177
- [“Deleting a Node in the Tree View”](#) on page 178
- [“Moving a Node in the Tree View”](#) on page 178
- [“Changing the Name or Value of a Node in the Tree View”](#) on page 178

- [“Obtaining the XPath for a Node”](#) on page 179

## 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.

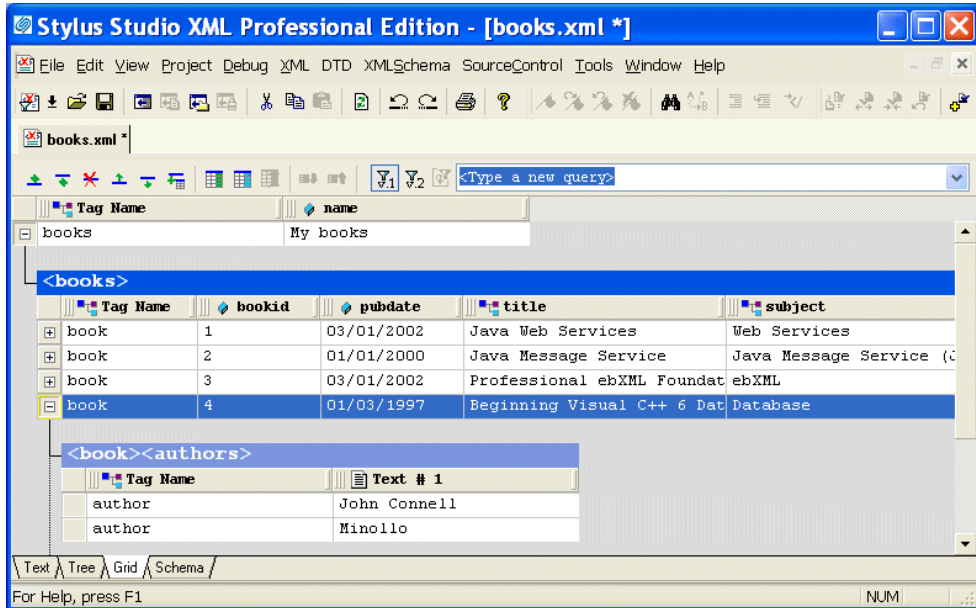
## Updating XML Documents Using the Grid Tab



The **Grid** view of an XML document is useful for structured data – it is a convenient way

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

to view and work with documents that contain multiple instances of the same type of element.



**Figure 126. 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:

- “Features of the Grid Tab” on page 181
- “Moving Around the Grid Tab” on page 185
- “Working with Rows” on page 187
- “Working with Columns” on page 189
- “Working with Tables” on page 191

Stylus Studio provides a video demonstration of Stylus Studio’s XML Editor **Grid** tab. Visit our Web site to view this and other Stylus Studio video demonstrations:

[http://www.StylusStudio.com/xml\\_videos.html](http://www.StylusStudio.com/xml_videos.html)

## Features of the Grid Tab

This section describes the layout and features of the **Grid** tab. It covers the following topics:

- [“Layout of the Grid Tab”](#) on page 181
- [“Expanding and Collapsing Nodes”](#) on page 182
- [“Collapsing Empty Nodes”](#) on page 182
- [“Renaming Nodes”](#) on page 184
- [“Resizing Columns”](#) on page 184
- [“Showing Row Tag Names”](#) on page 184

### Layout of the Grid Tab

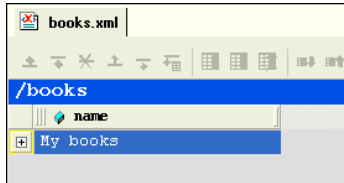
The **Grid** tab consists of a tool bar and a display area. The tool bar has buttons to perform actions and operations on both the grid itself and on the underlying XML document represented in the grid. An example of the former is the ability to show the child elements of the document’s root element; 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 229 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.

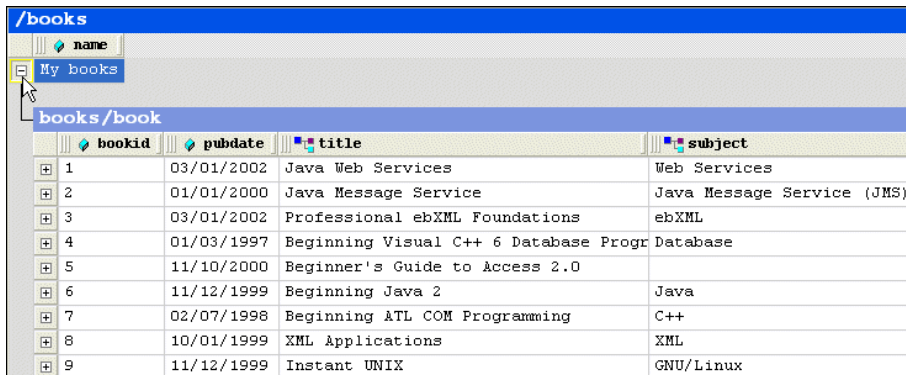
## Expanding and Collapsing Nodes

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 127](#).



**Figure 127. 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 128](#).



**Figure 128. 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 (⊕).

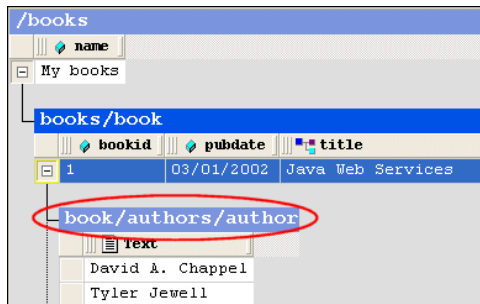
## Collapsing Empty Nodes

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:

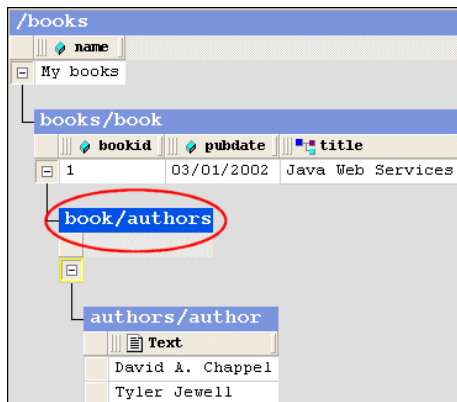
```
<authors>
  <author>David A. Chappel</author>
  <author>Tyler Jewell</author>
</authors>
```

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 129](#) shows the default display for the author element. Notice that the header, book/authors/author, contains information about the container node, authors.



**Figure 129. Table Headers Show Full Path**

If you want, however, you can display the tables associated with container nodes, as shown in [Figure 130](#).



**Figure 130. 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 131](#).



**Figure 131. 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 128, 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 132.

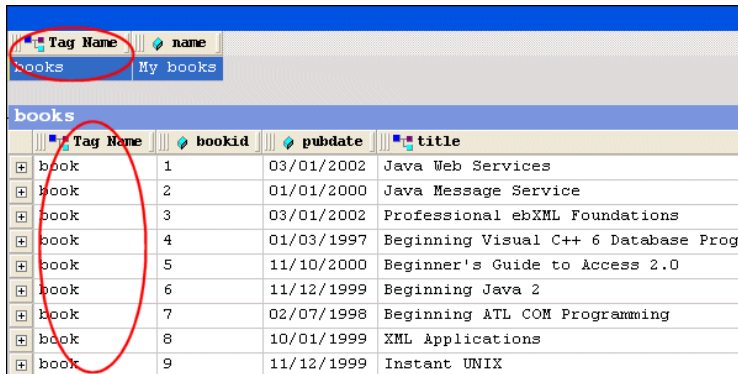


Figure 132. Displaying the Root's Child Element

- ◆ 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.
Page Down	Moves the row highlight to the last row in the document.

**Table 14. Keyboard Navigation in the Grid**

<b>Key</b>	<b>Action</b>
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.

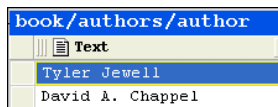
### How Grid Changes Affect the XML Document

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 133](#),



**Figure 133. 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 133](#), 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 188
- [“Adding and Deleting Rows”](#) on page 188

### 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 189
- “[Adding Columns](#)” on page 189
- “[Deleting Columns](#)” on page 190
- “[Reordering Columns](#)” on page 190
- “[Renaming Columns](#)” on page 191
- “[Changing a Value](#)” on page 191

### 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 134](#), 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 134. 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 190.

## Deleting Columns

### ◆ To delete a column:

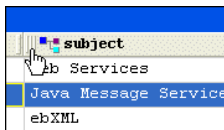
1. Select a cell in the column you want to delete.  
A yellow border appears around the cell you select.
2. Click **Delete Column** (  ). This action is also available from the **XML > Grid Editing** menu and from the grid shortcut menu. The column is deleted from the table.

## 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 135. 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:**

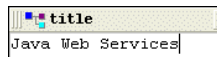
1. Select a cell in the column you want to rename.  
A yellow border appears around the cell you select.
2. Click **Rename Column** (📄). This action is also available from the **XML > Grid Editing** menu and from the column shortcut menu.  
The column is renamed.

## 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 136. 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.

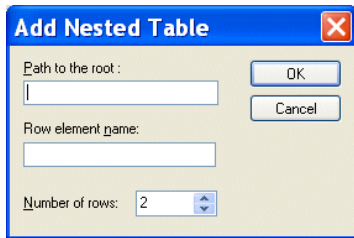
This section covers the following topics:

- [“Adding a Nested Table”](#) on page 192
- [“Moving a Nested Table”](#) on page 193
- [“Deleting a Table”](#) on page 193

- “[Sorting a Table](#)” on page 194
- “[Copying a Table as Tab-Delimited Text](#)” on page 194

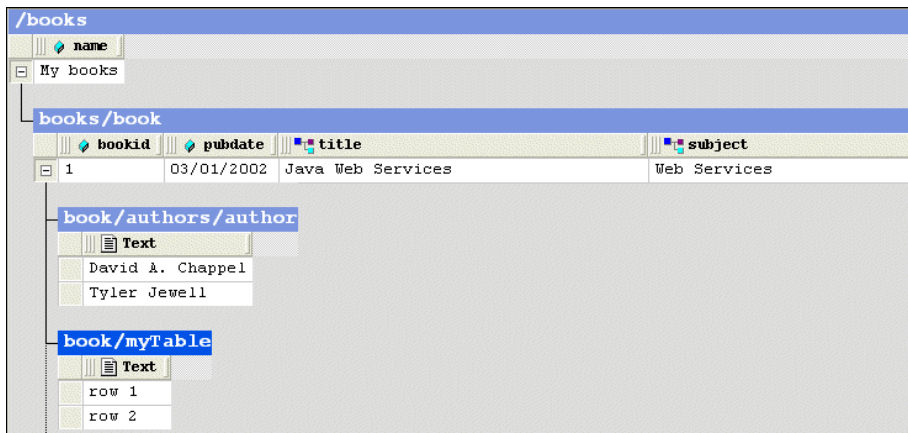
### Adding a Nested Table

You add nested tables to a document in the **Grid** tab using the **Add Nested Table** dialog box, shown in [Figure 137](#). 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 137. Add Table Dialog Box**

A nested table is created as a child of the current element. The nested table shown in [Figure 138](#), myTable, was created as a child of the <book> element.




**Figure 138. 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.

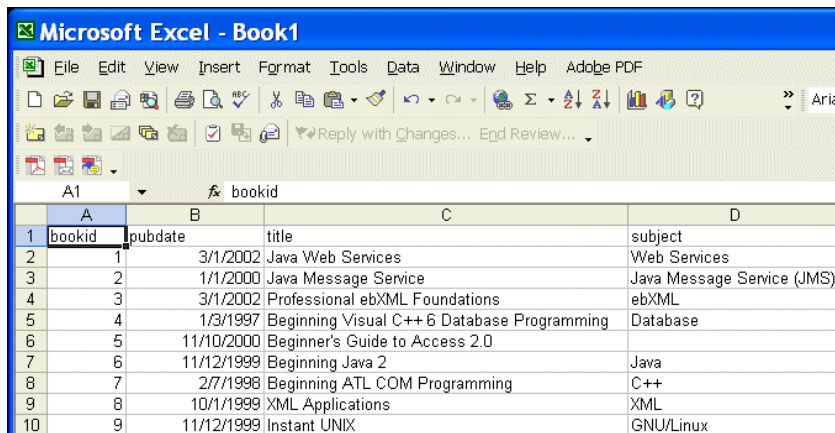
**Tip**

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 139](#) shows books/book in books.xml pasted into Microsoft Excel, for example.



**Figure 139. 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 Edition and Stylus Studio XML Professional Edition.

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.

Stylus Studio provides a video demonstration of Stylus Studio's XML Diff tool. Visit our Web site to view this and other Stylus Studio video demonstrations:

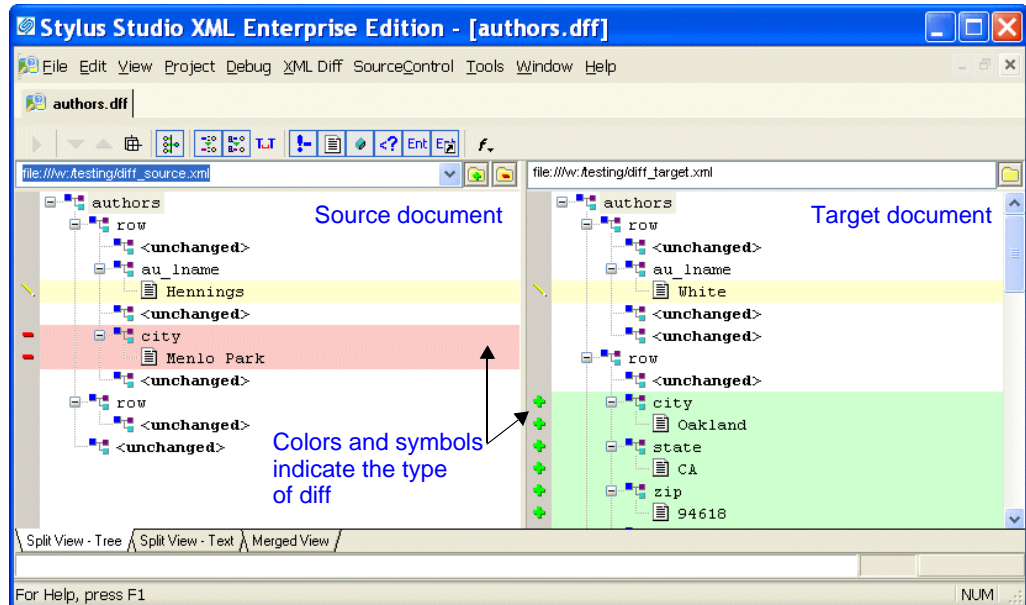
[http://www.StylusStudio.com/xml\\_videos.html](http://www.StylusStudio.com/xml_videos.html)

This section covers the following topics:

- “Overview” on page 196
- “Diffing Folders” on page 201
- “The XML Diff Viewer” on page 205
- “Diffing a Pair of XML Documents” on page 213
- “Diffing Multiple Documents” on page 214
- “Modifying Default Diff Settings” on page 219
- “Running the Diff Tool from the Command Line” on page 223

## 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 140](#)), or the contents of two folders (as shown in [Figure 144](#)).



**Figure 140. 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 197
- “The Diff Configuration File” on page 197
- “What Diffs Are Calculated?” on page 197
- “Tuning the Diffing Algorithm” on page 198
- “When Does the Diff Run?” on page 199

- [“Running the Diff Manually”](#) on page 200
- [“Symbols and Background Colors”](#) on page 200

### 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 199 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)

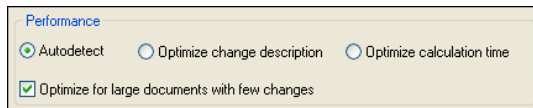
See “[Modifying Default Diff Settings](#)” on page 219 to learn how to set these and other diff options.

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 201 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 141. 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 219 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 199, 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 223.

### 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 142. 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 143. Expanded Node with Changes**

**Tip** Hover the mouse point over these symbols to display tool tips that describe the nature of the change.

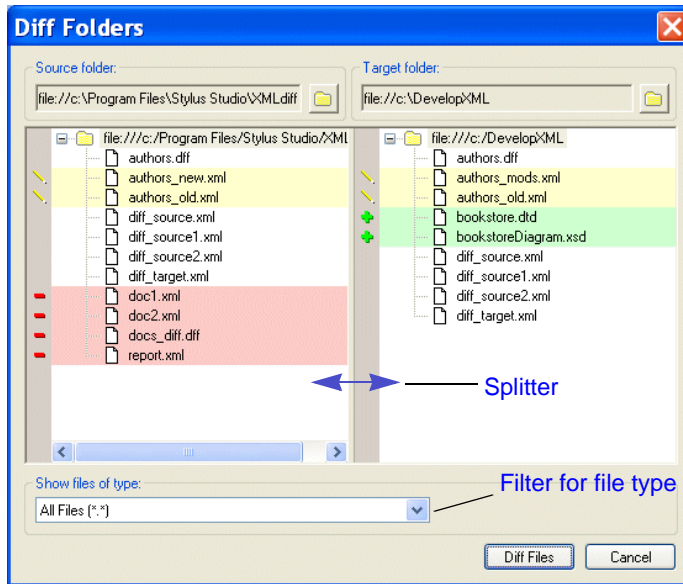
## Additional Symbols for Diffing Multiple Sources

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 215.

## Diffing Folders

Stylus Studio allows you to diff two folders. As shown in [Figure 144](#), the **Diff Folders** dialog box displays the contents of each folder; symbols and colors, described in

“Symbols and Background Colors” on page 200, identify the types of changes in the respective folders.



**Figure 144. The Diff Folders Dialog Box After a Diff**

This section covers the following topics:

- “Features” on page 202
- “How to Diff Folders” on page 203
- “How to Diff Documents from the Diff Folders Dialog Box” on page 205

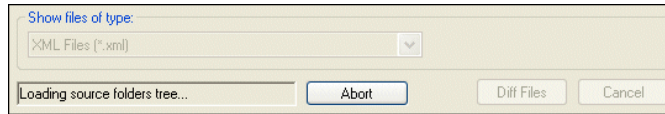
## Features

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 144](#).
- 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 145. Abort Button Lets You Cancel Long Load Operations**

### Tip

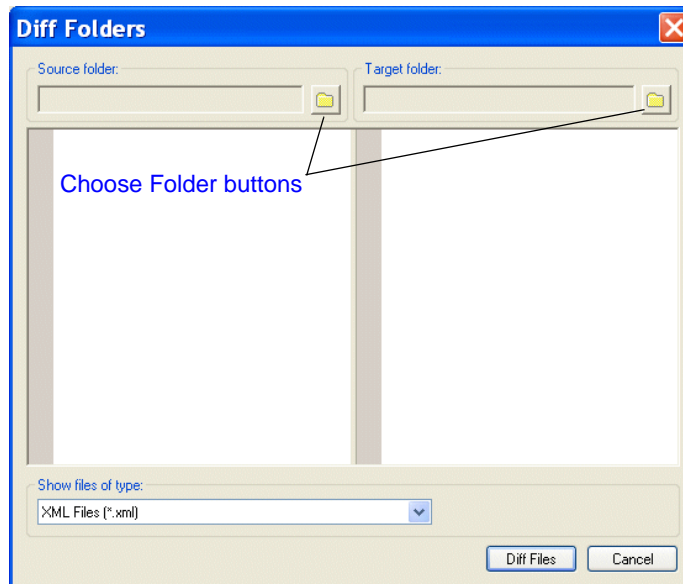
This feature is also part of the XML Diff Viewer.

- 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 205 for more information on this topic.

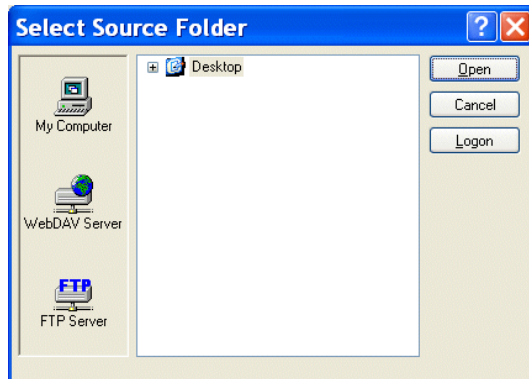
## How to Diff Folders

### ◆ 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.)

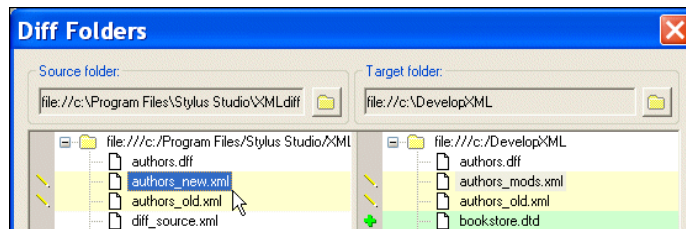
## How to Diff Documents from the Diff Folders Dialog Box

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 213.

## 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.

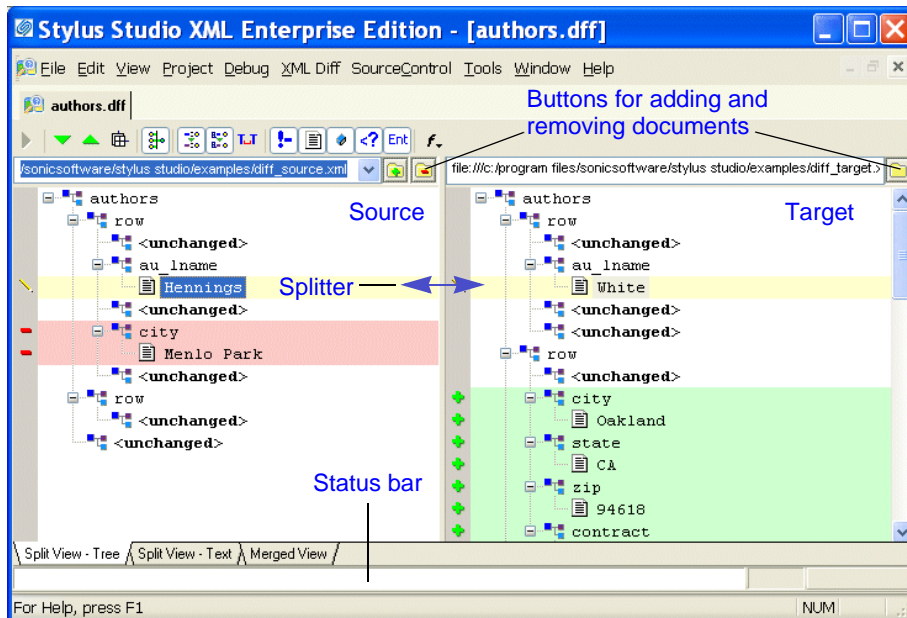
This section covers the following topics:

- [“Split View - Tree”](#) on page 206
- [“Split View - Text”](#) on page 207
- [“Merged View”](#) on page 208

- “View Symbols and Colors” on page 209
- “The XML Diff Viewer Tool Bar” on page 209
- “Tools for Working with Documents” on page 212

### 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 146, shows the documents side-by-side using a tree/node representation.



**Figure 146. 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.

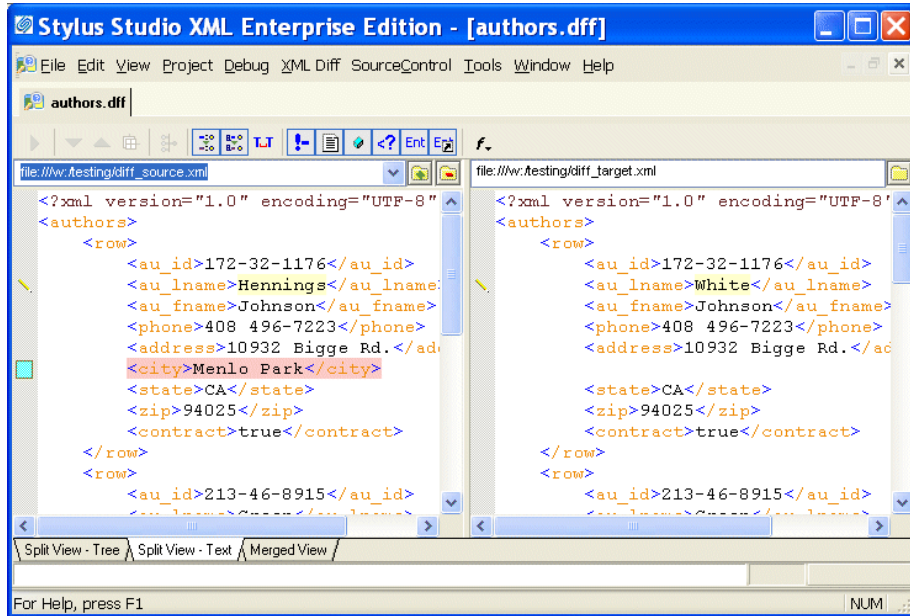
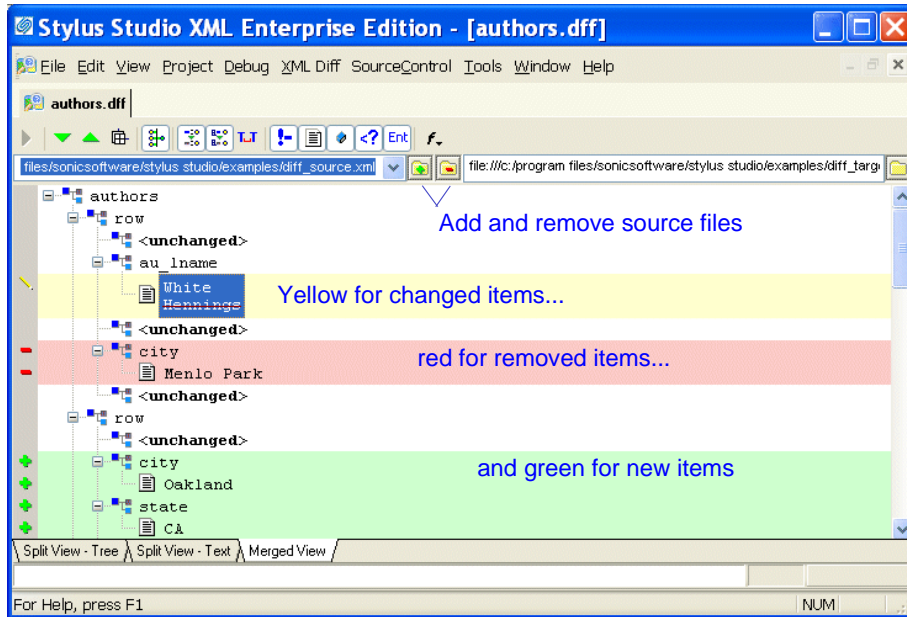


Figure 147. XML Diff Viewer – Split View - Text

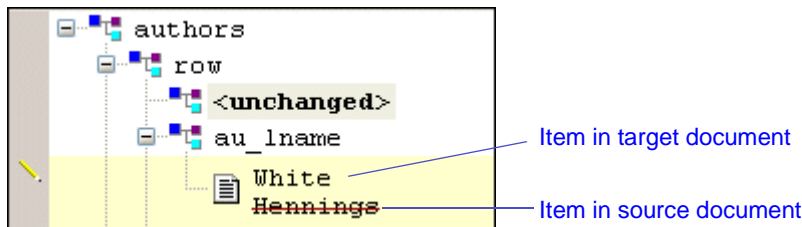
## Merged View

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 148](#).



**Figure 148. 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 149](#).



**Figure 149. 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

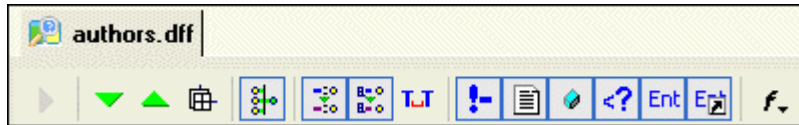
All views use the same symbols and color schemes to identify the types of changes detected by the Stylus Studio diff calculation – by default, green for added items, yellow for changed items, and red for removed items. In addition, the text font and size are controlled by the settings for the XML Editor on the **Editor Format** page of the **Options** dialog box.

See “[Symbols and Background Colors](#)” on page 200 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 150](#), 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 150. 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 <b>On changes</b> and <b>If files modified</b> settings. See “ <a href="#">Engine Settings</a> ” on page 221.
	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**













<b>Tool Button</b>	<b>Description</b>
	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 <b>Options</b> . page. See “ <a href="#">Engine Settings</a> ” on page 221.
	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 <b>Options</b> . page. See “ <a href="#">Engine Settings</a> ” on page 221.
	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 <b>Options</b> . page. See “ <a href="#">Engine Settings</a> ” on page 221. 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 <b>Options</b> . page. See “ <a href="#">Engine Settings</a> ” on page 221. 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 <b>Options</b> . page. See “ <a href="#">Engine Settings</a> ” on page 221.
	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 <b>Options</b> . page. See “ <a href="#">Engine Settings</a> ” on page 221. 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 <b>Options</b> . page. See “ <a href="#">Engine Settings</a> ” on page 221. Note that this feature affects only the display, and not the calculation, of text block differences.



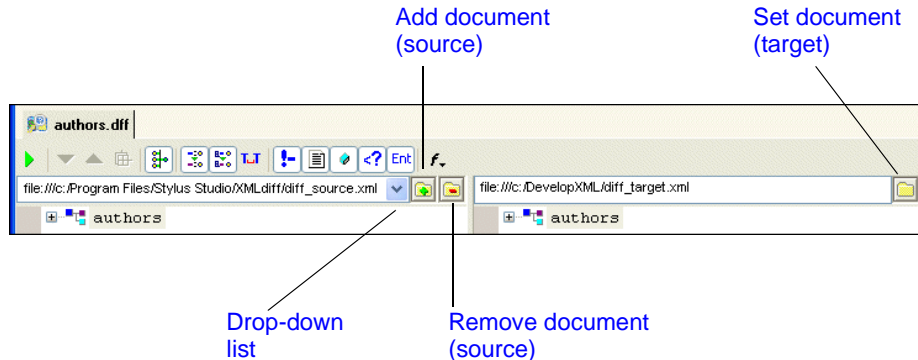
Table 16. XML Diff Tool Bar Buttons

<b>Tool Button</b>	<b>Description</b>
	<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 <b>Options</b> page. See “<a href="#">Engine Settings</a>” on page 221.</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 <b>Options</b> page. See “<a href="#">Engine Settings</a>” on page 221.</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 <b>Options</b> page. See “<a href="#">Engine Settings</a>” on page 221.</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 <b>Options</b> page. See “<a href="#">Engine Settings</a>” on page 221.</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:



**Figure 151. Buttons for Adding and Removing Documents**

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 215 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 196, which describes basic information about the Stylus Studio Diff tool, and “[The XML Diff Viewer](#)” on page 205, 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 196, which describes basic information about the Stylus Studio Diff tool, and [“The XML Diff Viewer”](#) on page 205, which describes features of the XML Diff Viewer and how to use them.

This section covers the following topics:

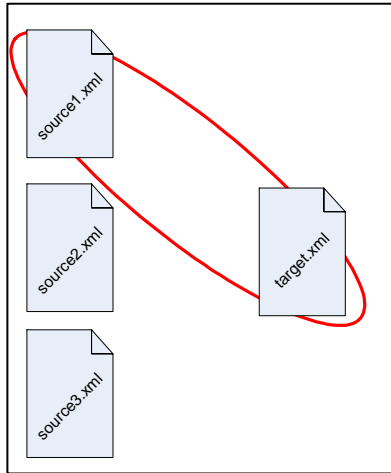
- [“Document Focus”](#) on page 214
- [“Symbols Used in the Target Document Window”](#) on page 215

#### 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 152. 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.

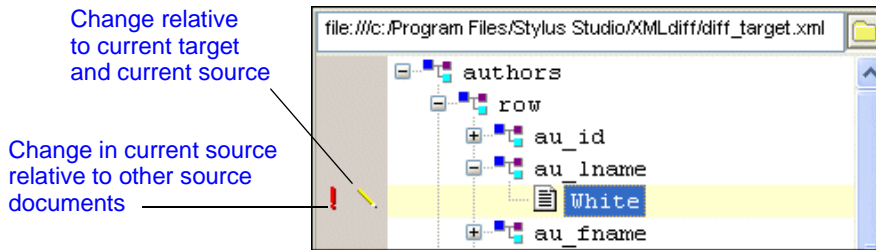
When a source document has focus:

- 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 200, indicate the ways in which the change identified in the

current source document differs from changes to the same node in other source documents.

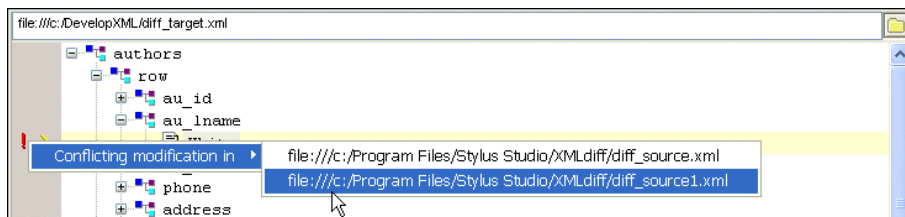


**Figure 153. Example of Symbols Used When Diffing Multiple Documents**

As shown in [Figure 153](#), 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 154](#), 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 154. Easily Navigate to Changed Nodes in Other Source Documents**

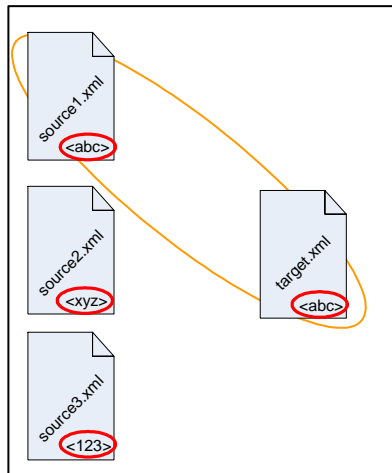
Clicking on a document in this menu changes the current focus to that source document, allowing you to easily navigate to the same node in a different document.

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**

<b>Symbol</b>	<b>Description</b>	<b>Meaning</b>
	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 155](#), which illustrates diffing three documents. In this example, the node in question is circled in red.







**Figure 155. 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 155 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.



- 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.

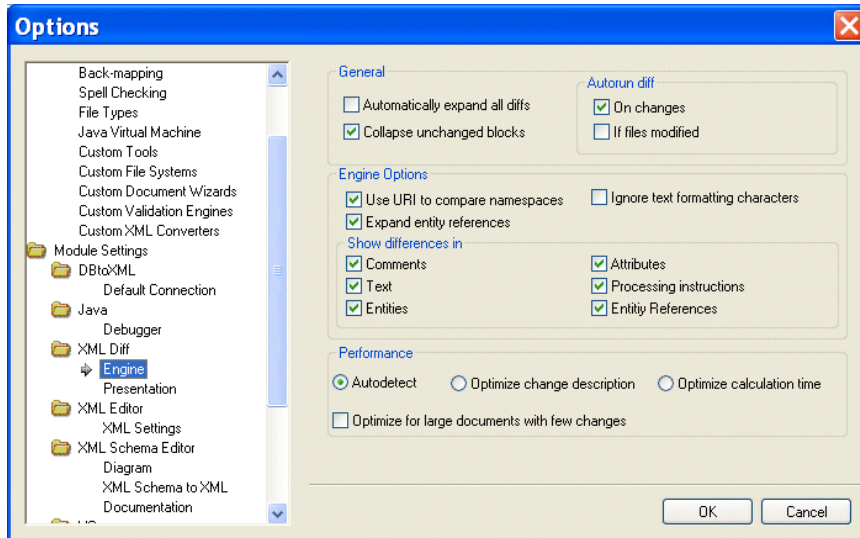
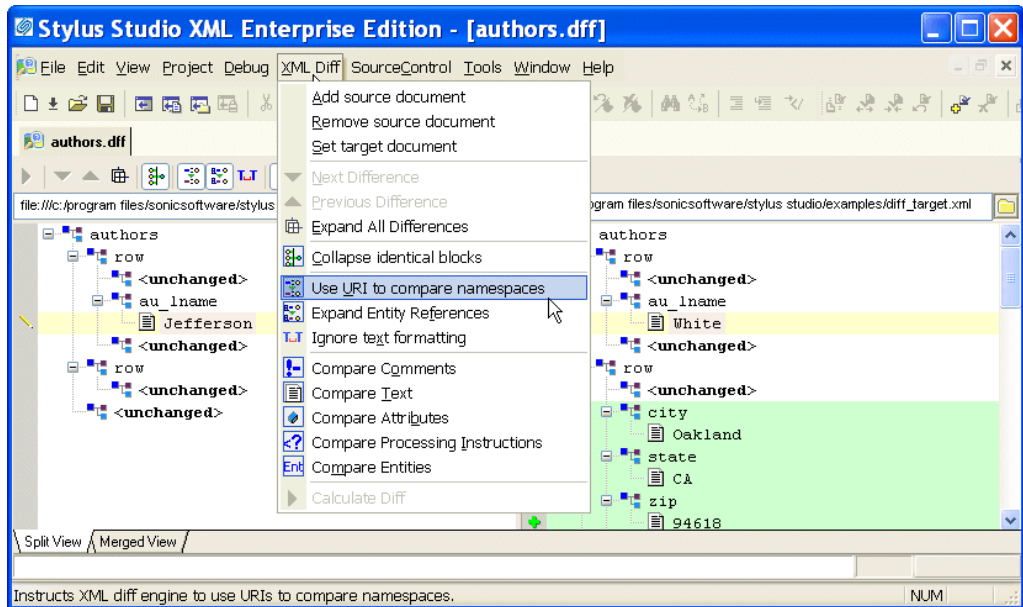


Figure 156. Engine Options Page for XML Diff

Settings on the **Engine** page are reflected in the Diff editor tool bar, and in the **XML Diff** menu, shown here.



**Figure 157. 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 220
- “Engine Settings” on page 221
- “Presentation Options” on page 223

### 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 199 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

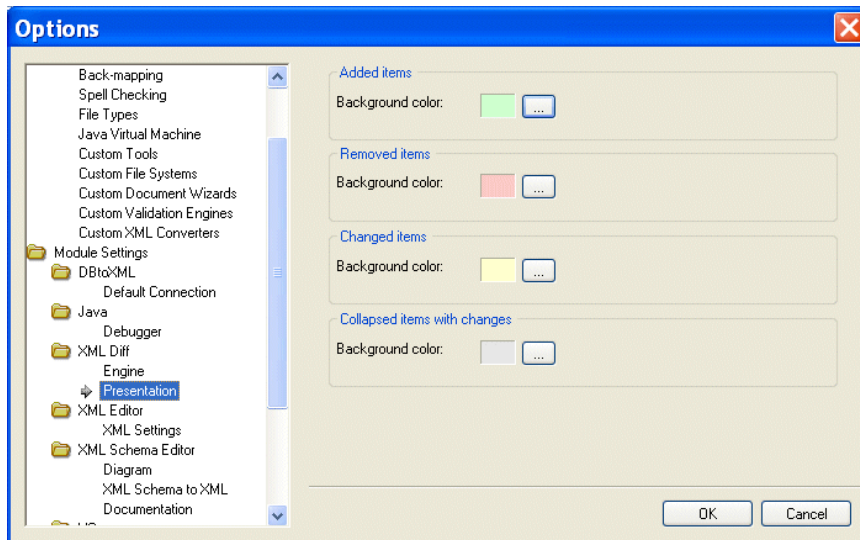
Diffing large, numerous, or complex documents can be time-consuming. Stylus Studio provides controls that let you choose algorithm tunings that have been optimized for change description or calculation time.

- **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 158.** 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 221.

**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**

<b>Parameter</b>	<b>Description</b>
-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 227.

This section discusses the following topics:

- [“Associating an External Schema With a Document”](#) on page 226
- [“Having Stylus Studio Generate a Schema”](#) on page 227
- [“Validating XML Documents”](#) on page 227
- [“Updating a Document’s Schema”](#) on page 228
- [“Removing the Association Between a Document and a Schema”](#) on page 228

## 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 110.
  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 228.

## 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 589. Instructions for updating an XML Schema are in [“Creating an XML Schema in Stylus Studio”](#) on page 498. 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

In each view of an XML document, you can query the document to obtain a subset of the information in that document. You can query stylesheets and XML Schema documents, but you cannot query DTD schema documents because they are not XML. This section discusses the following topics:


- [“Steps for Querying a Document”](#) on page 229
- [“Displaying Query Results”](#) on page 230
- [“Saving Query Results”](#) on page 230

### Steps for Querying a Document

◆ **To query a document:**

1. Open the document you want to query.
2. In the document window tool bar, click in the query field. This field contains either a prompt to create a new query, or it contains a query you ran previously. For information about how to write a query, see [“Writing XPath Expressions”](#) on page 613.

*Alternative:* Press F6.

3. Press Enter to run the query or click **Refresh Query** , which is to the left of the query field at the top of the document window.

If a document is part of a Stylus Studio project, Stylus Studio saves the queries you specify. If you save, close, and reopen a document, the queries you already specified are still defined. In the document window, click the down arrow to the right of the query field to display the defined queries.

Stylus Studio assigns a number, beginning with 1, to each query you define. This allows you to run multiple queries against the same document and distinguish one from another.

After you make some modifications to your document, you might want to run a query that you already ran. Display the query in the field in the document tool bar and click **Refresh Query**.

## Displaying Query Results

How Stylus Studio displays the result of a query depends on the type of result.

- If the XPath processor returns a Boolean, numeric, or string value, Stylus Studio displays the value in the **Query Output** window.
- If the XPath processor returns a collection of nodes, Stylus Studio displays the nodes in a DOM tree structure in the **Query Output** window to the right of the main XML editor pane. The number of **Hits** is the number of top-level nodes the query returned. You can expand and collapse the elements as needed. If you click a node in the result, then in the XML document window Stylus Studio moves the focus to the source node for the result node you clicked.

A tab at the bottom of the **Query Output** window displays the number that Stylus Studio assigned to your query. After you run multiple queries on the same document, there is a tab for each query that you run. This allows you to view the results of the different queries as needed.

## Saving Query Results

After you execute a query, Stylus Studio displays the results in the **Query Output** window.

### ◆ To save query results:

1. In the **Query Output** window, click **Save Result** .


2. In the **Save As** dialog box that appears, navigate to the directory that you want to contain the query result.
3. In the **File name:** field, type the name for the file that will contain the query result.
4. Click **Save**.

To add a query result file to a project, see [“Adding Files to Projects”](#) on page 122.

## 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 484.

### 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


### Learning About Regular Expressions

To learn more about regular expression syntax, visit <http://www.boost.org/libs/regex/doc/syntax.html>.

## 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 Sleepycat Software's Berkeley DB XML), 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**

Stylus Studio uses *adapters* to convert an incoming stream of data from a native format to an outgoing stream of XML, or vice-versa. You can create your own adapters, using the *Convert to XML* module, or you can use one of the many built-in adapters bundled with Stylus Studio.

This chapter describes how to use Convert to XML to create an adapter, how to use files you convert on-the-fly elsewhere in Stylus Studio – as a source for XQuery and XSLT design, for example – and how to use Stylus Studio’s built-in adapters.



Convert to XML is available only in Stylus Studio XML Enterprise Edition and Stylus Studio XML Professional Edition.

For more information

Stylus Studio provides a video demonstration of Convert to XML and adapter usage. Visit our Web site to view this and other Stylus Studio video demonstrations:

[http://www.StylusStudio.com/xml\\_videos.html](http://www.StylusStudio.com/xml_videos.html)

Contents

This chapter covers the following topics:

- “Overview of Convert to XML” on page 236
- “Choosing an Input File” on page 239
- “The Convert to XML Editor” on page 240
- “Parts of an Input File” on page 251
- “Working with Regions” on page 253
- “Working with Fields” on page 259
- “Controlling XML Output” on page 267
- “Creating an Adapter” on page 278
- “Using Adapters in Stylus Studio” on page 280

- [“More About Converting EDI”](#) on page 284
- [“Invoking an Adapter Programmatically”](#) on page 287
- [“User-Defined Adapter Properties Reference”](#) on page 298

## Overview of Convert to XML

The Convert to XML module consists of

- An *adapter engine* that processes non-XML source files using the adapter configuration file (.conv) you specify. The adapter engine is called silently any time you open a file using a Convert to XML adapter. You can also run the adapter engine manually to test the results of the adapters you create, but it has no user-visible or configurable components.
- An *editor* that lets you configure the instructions in the .conv file used by the adapter engine to convert non-XML files to XML. The Convert to XML Editor, shown in [Figure 159](#), displays information read from the input file you are using as a model to build your adapter, as well as values you want the adapter engine to use when

converting other files of this type to XML. The Convert to XML Editor is displayed automatically when you create a new adapter or open an existing one.

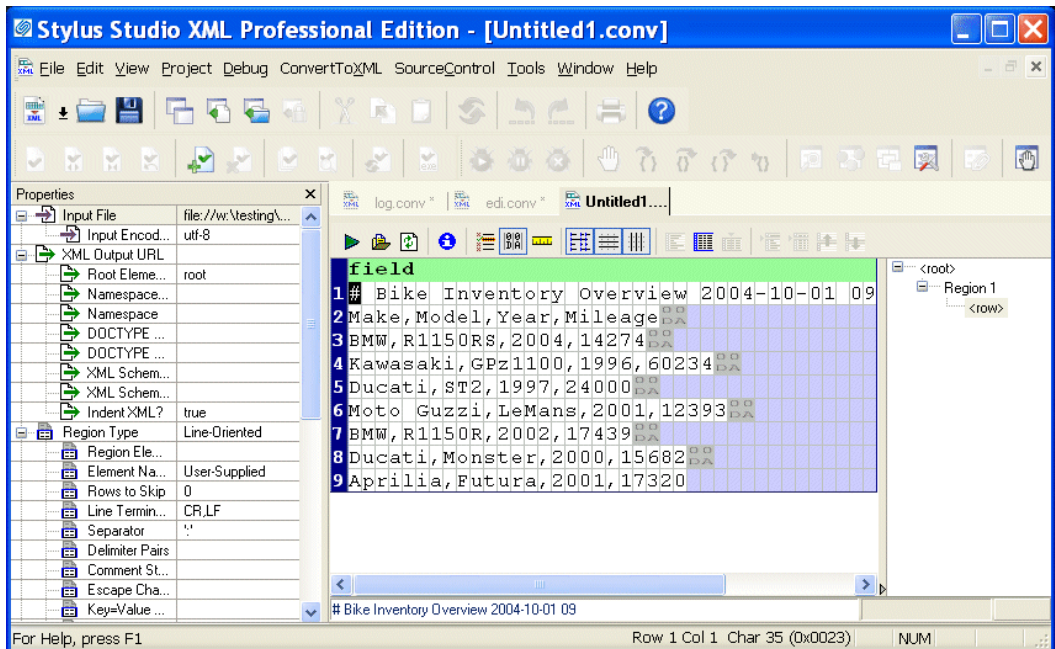


Figure 159. Convert to XML Editor

## File Support

Stylus Studio's Convert to XML module makes it easy to build adapters for 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, layout of the input file, or you can specify these settings manually, as shown in [Figure 160](#).

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.

### Using Convert to XML

Generally speaking, you use Convert to XML to create an adapter (also known as a *converter*), and then you use that adapter to convert one or more non-XML files to XML. The process of using Convert to XML involves the following steps:

1. Create the adapter:
  - a. Select **File > New > Convert to XML** from the Stylus Studio menu to display the **New Converter** dialog box.
  - b. Select the file you want to convert to XML and open it in the Convert to XML Editor.
  - c. Review the default adapter configuration settings displayed in the Convert to XML Editor, and modify as required.
  - d. Run the adapter and review the conversion results displayed in the **Preview** window.
  - e. Save the adapter (and, optionally, the XML output for the input file selected in [step b](#)).
2. Use the adapter to open a new non-XML file of the same type as the file you used to define the adapter.

The Convert to XML adapter engine converts the new file to an XML document, displaying it in the Stylus Studio editor you specify when opening the file.

The steps in this process are described in greater detail in the following sections.

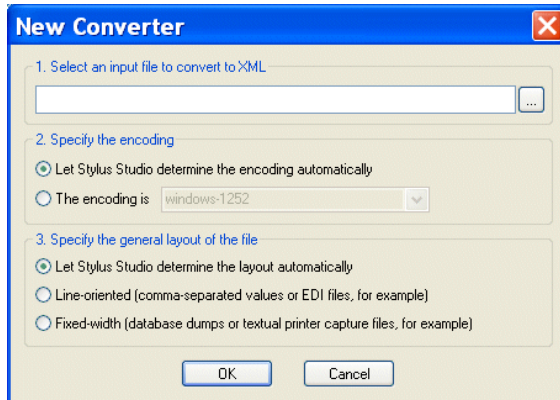
### Other Ways to Convert Files to XML

In addition to creating your own adapter, you can convert files to XML

- Using Stylus Studio's built-in adapters – Stylus Studio comes bundled with built-in adapters for numerous file formats, including EDI, CSV, dBase, and Windows .ini files. See [“Built-In Adapters”](#) on page 281 for more information on this topic.
- Using Stylus Studio document wizards – Stylus Studio document wizards help you convert CSV and fixed-width files to XML. See [“Converting Text Files to XML Documents”](#) on page 156.


## Choosing an Input File

The first step of creating an adapter is to select an input file.



**Figure 160. Specify Input File Type Manually or Let Stylus Studio Decide**

The input file can be any type. If you are planning to use the adapter 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 adapter 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.

**Note** File systems that return only XML files, such as Sleepycat Software’s Berkeley DB XML, do not appear when you browse for input files using the **Open** dialog box()

# The Convert to XML Editor

You use the Convert to XML Editor, shown here, to build an adapter. The Convert to XML Editor appears when you create a new adapter, or open an existing one (a .conv file).

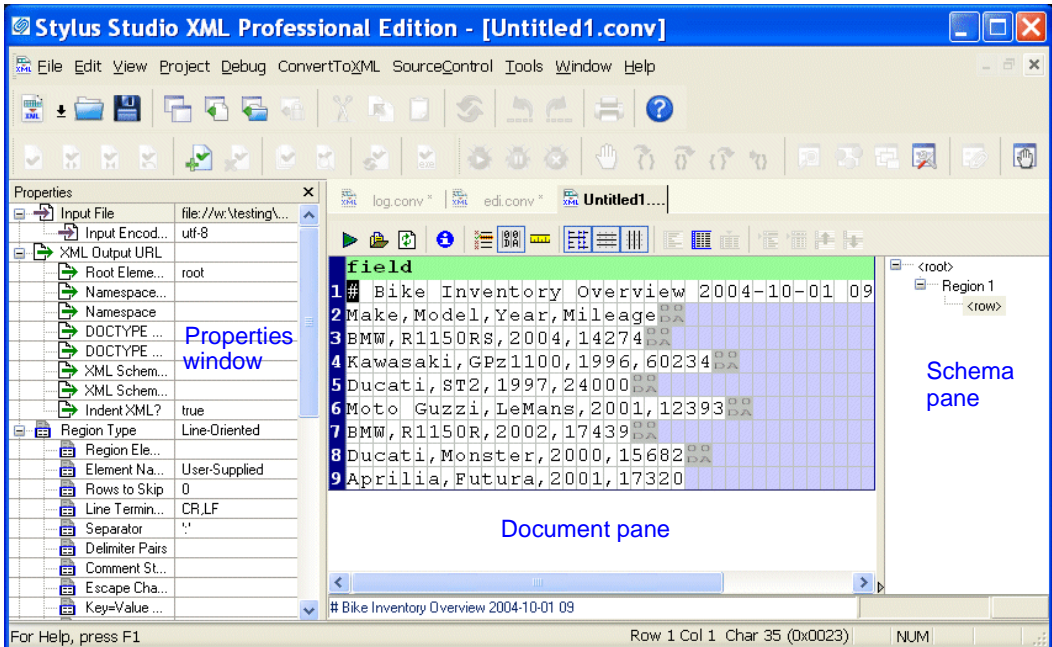


Figure 161. Convert to XML Editor

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 Convert to XML Editor, including how it interacts with the adapter engine. This section covers the following topics:

- “**Document Pane**” on page 241
- “**Properties Window**” on page 248
- “**Schema Pane**” on page 250

## 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 241
- [“Display of Delimiting and Control Characters”](#) on page 242
- [“Field Names”](#) on page 243
- [“Document Pane Display Features”](#) on page 244
- [“Moving Around the Document”](#) on page 246

### 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
```

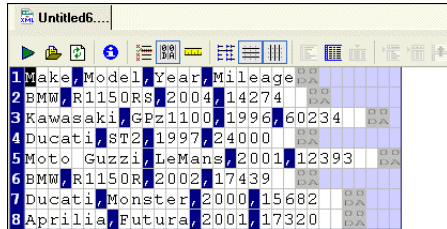
[Figure 162](#) shows how this character-separated input file appears in the Convert to XML Editor's document pane. By default, Stylus Studio aligns columns and fills the empty cells of the shorter rows with a light blue to aid readability:

	field	field	field	field	
1	Make	Model	Year	Mileage	00
2	BMW	R1150RS	2004	14274	00
3	Kawasaki	GPz1100	1996	60234	00
4	Ducati	ST2	1997	24000	00
5	Moto Guzzi	LeMans	2001	12393	00
6	BMW	R1150R	2002	17439	00
7	Ducati	Monster	2000	15682	00
8	Aprilia	Futura	2001	17320	00

**Figure 162. Character-Separated File in the Convert to XML Document Pane**

## Converting Non-XML Files to XML

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 **Convert to XML > Align Fields** on the menu. This results in the layout shown in [Figure 163](#).

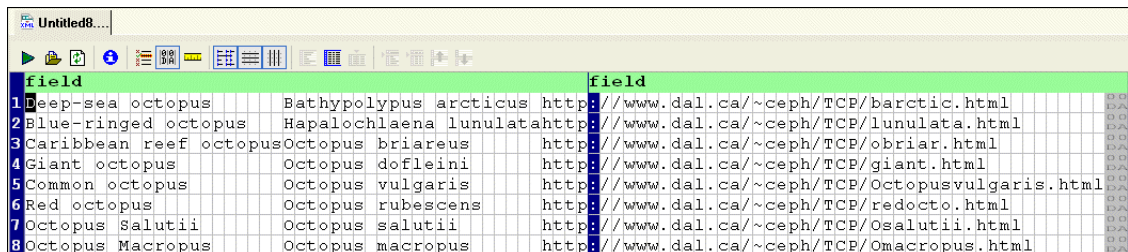


**Figure 163. Character-Separated File without Aligned Fields**

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/redocto.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:





**Figure 164. Fixed-Width File in the Convert to XML Document Pane**

## 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 162](#), 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 311.

- 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:

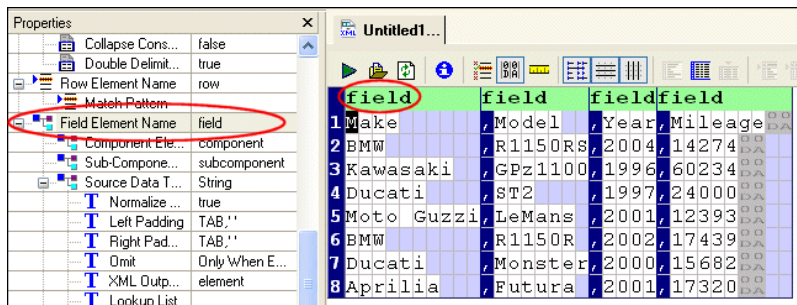


Figure 165. User-defined Field Names are Shown in Green

## Converting Non-XML Files to XML

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.

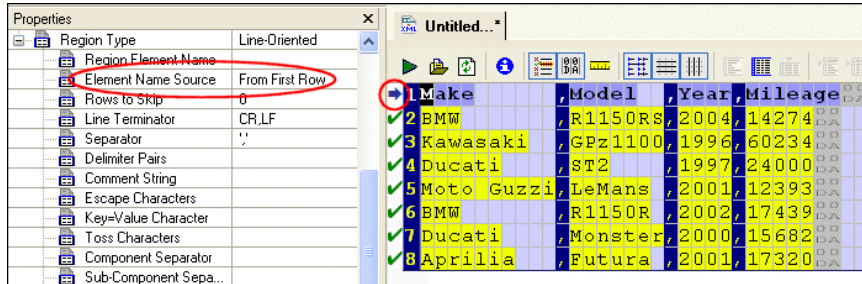


Figure 166. Blue Arrow Indicates Field Names Taken from the File

See “[Naming Fields](#)” on page 260 to learn more about naming fields for XML output by Convert to XML converters.

## Document Pane Display Features

In addition to aligning fields in character-delimited files, the Convert to XML Editor’s document pane has several other features that aid readability.

### Ruler

You can display a ruler that identifies each column:

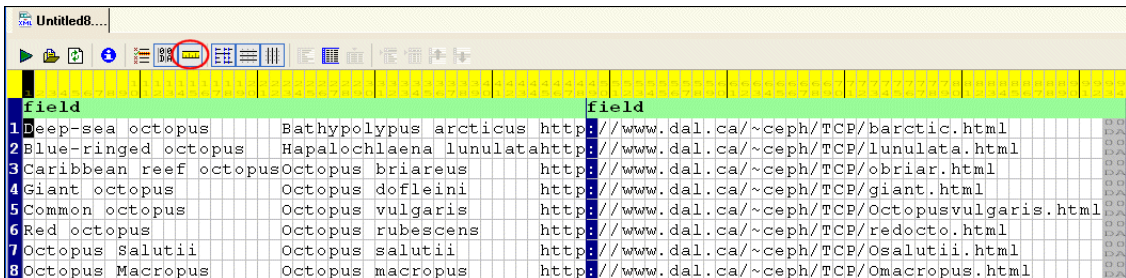

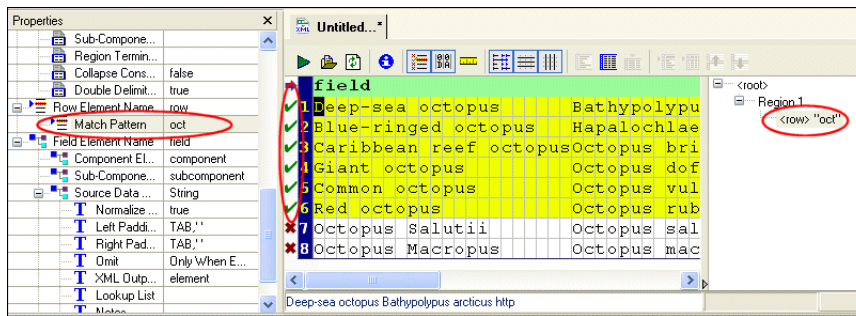


Figure 167. Ruler Helps Identify Columns

To display the ruler, click the **Toggle Ruler** button () on the tool bar, or select **ConvertToXML > 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 168. Matching Rows Are Displayed in Yellow**

To highlight matching rows, click the **Highlight Matching Rows** button () on the tool bar, or select **ConvertToXML > Highlight Matching Rows** from the menu.

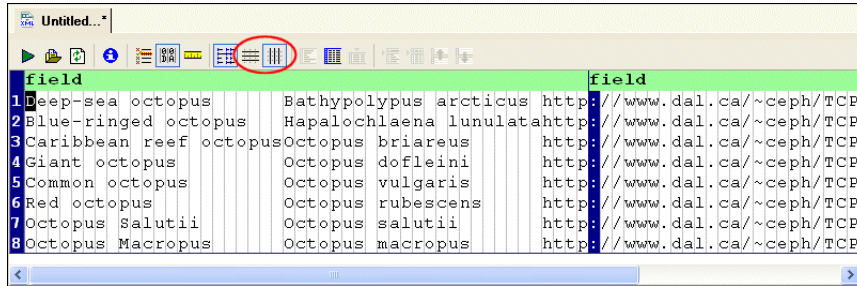
Matching rows are displayed in a light yellow, with a green check in the pane's margin. A red X identifies rows that do not match the pattern. Gray squares identify rows that match a pattern other than the pattern defined for the row that currently has focus. See [“Specifying Multiple Match Patterns”](#) on page 272 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 270 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 169](#), horizontal lines are hidden from the display:



**Figure 169. 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

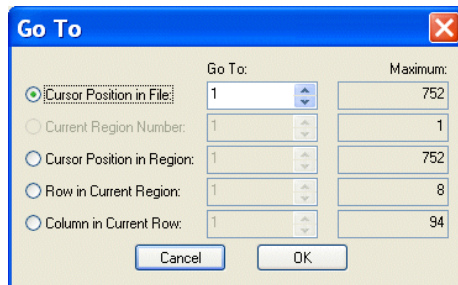
By default, the input document is displayed using the Courier New font in 12pt. You can change the display font to suit your personal preference using the **Edit > Change Font** and **Edit > Font Size** menus.

### Moving Around the Document

You can move the cursor around the document

- Using the Space bar on your keyboard
- Using the directional arrows and Page Up, Page Down, Home, and End keys on your keyboard
- Using your mouse (click on the character on which you want to place the cursor)

- Using the **Go To** dialog box



**Figure 170. 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 converter. You can use it to move the cursor to a specific

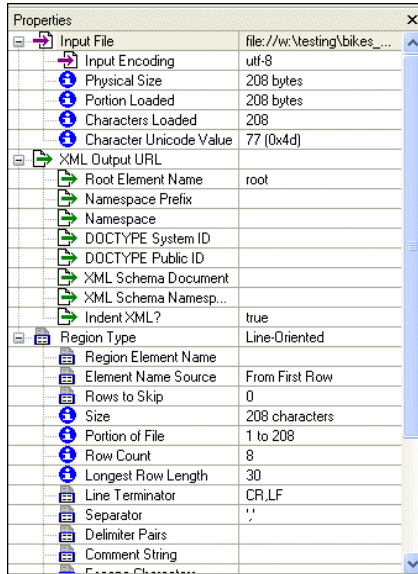
- Position in the file
- Region
- Position or row within a region
- Column within the current row

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 Convert to XML 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 171](#), displays information about the input file, as well as settings that Stylus Studio will use convert files to XML.








**Figure 171. 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 Convert to XML 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 Convert to XML converter 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 Convert to XML converter, 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: .

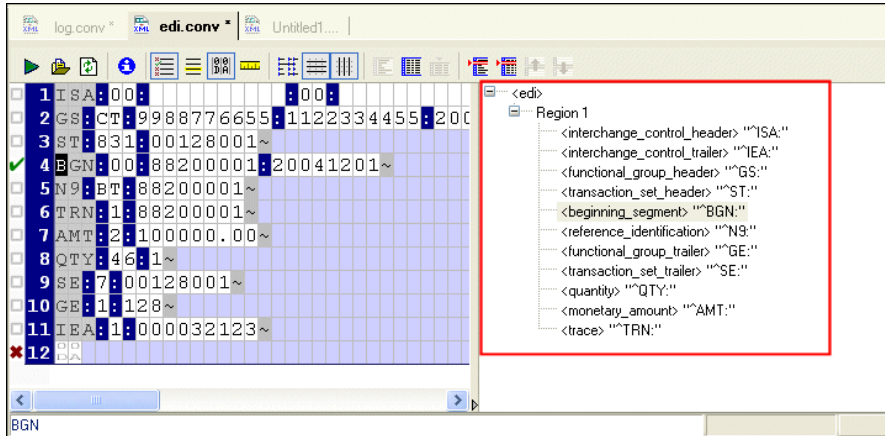
**Note** Informational properties, that is, properties that do not affect XML output, are displayed with the following icon: . These properties are displayed when you click the **Toggle Informational Properties** button.

## 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 “[User-Defined Adapter Properties Reference](#)” on page 298 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 172. 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 173](#). This functionality provides an alternative to editing the row name and specifying a match pattern in the **Properties** window.



**Figure 173. 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 252 to learn more about specifying conversion properties for rows.



## Parts of an Input File

Input files displayed in the Convert to XML 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:

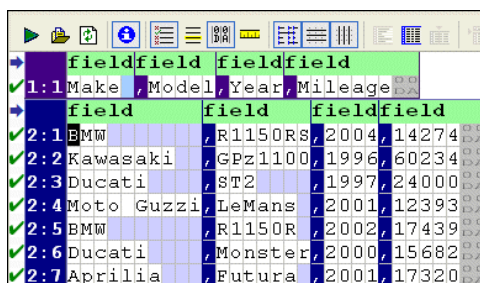
- “Regions” on page 251
- “Rows” on page 252
- “Fields” on page 253

### 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 Convert to XML Editor, and you can define your own.

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.

In the Convert to XML 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:



field	field	field	field
1:1 Make	, Model	, Year	, Mileage
field	field	field	field
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 174. 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 **Unknown**. Regions that are marked as unknown are grayed out in the Convert to XML 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, and mark a region so that it is excluded from output. For information on these and other topics, see [“Working with Regions”](#) on page 253.

### 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 269 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 Convert to XML Editor. See [“Adjusting Fixed-Width Regions”](#) on page 256 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 259 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 263 for more information.

## Component and Sub-Component Fields

Some file formats, including many EDI variants, allow fields to be subdivided into arrays, sub-fields, or composite fields. Collectively, these fields are referred to as *component fields* in Convert to XML, 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 253
- “[Adjusting Fixed-Width Regions](#)” on page 256
- “[Defining and Joining Regions](#)” on page 257
- “[Controlling Region Output](#)” on page 259

## 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 **Unknown**.

**Tip** Information about unknown regions is not displayed in the **Properties** window.

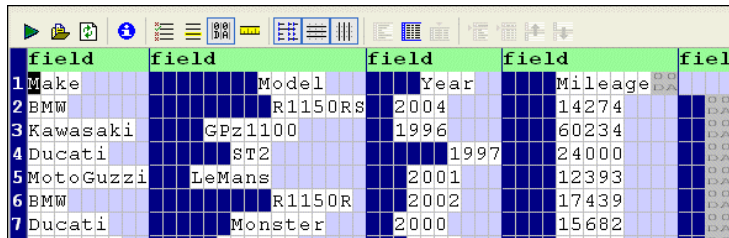
## Converting Non-XML Files to XML

Stylus Studio's Convert to XML interprets regions with CR/LF control characters 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

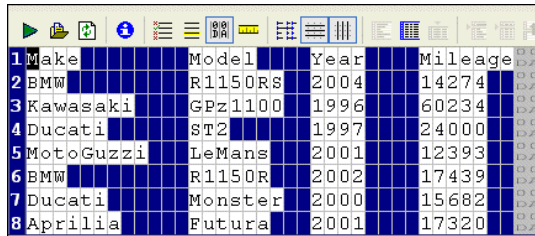
It is a simple .txt file with fields (Make, Model, and so on) that have been created using spaces. Each new row was created using the Enter key in the text editor, resulting in CR/LF control characters at the end of each line that cause Convert to XML to interpret the file as a single line-oriented region, like this:



**Figure 175. 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

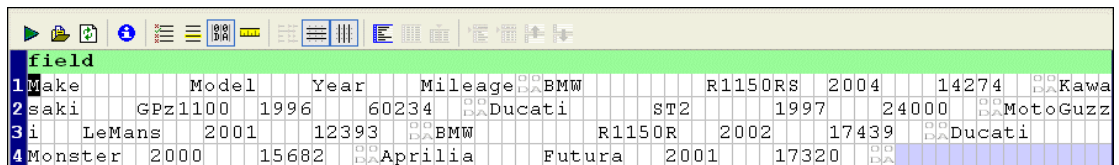
results in a *display* that resembles the source (Figure 176), but Stylus Studio still considers the region to be line-oriented.



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 176. Turning Off Align Fields Can Aid Readability**

When you convert a line-oriented region to a fixed-width region, Convert to XML 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 177.



1	field	Make	Model	Year	Mileage	BMW	Ducati	MotoGuzz
2	saki	GPz1100	1996	60234	Ducati	ST2	1997	24000
3	i	LeMans	2001	12393	BMW	R1150R	2002	17439
4	Monster	2000	15682	Aprilia	Futura	2001	17320	Ducati



**Figure 177. 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 256.

## How to Convert a Region Type

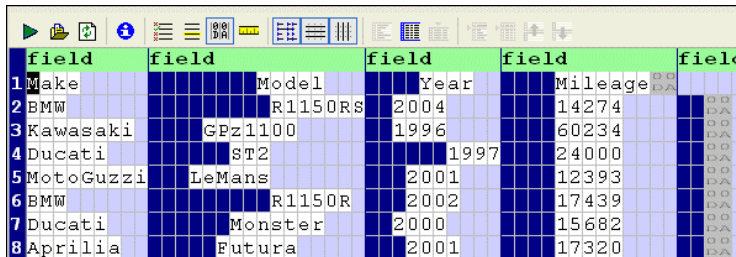
### ◆ To convert a region type:

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 **ConvertToXML** menu and the shortcut menu in the Convert to XML Editor.

- 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 256.

### Adjusting Fixed-Width Regions

Stylus Studio uses a default row length of 80 characters for fixed-width files. 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 178](#), 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

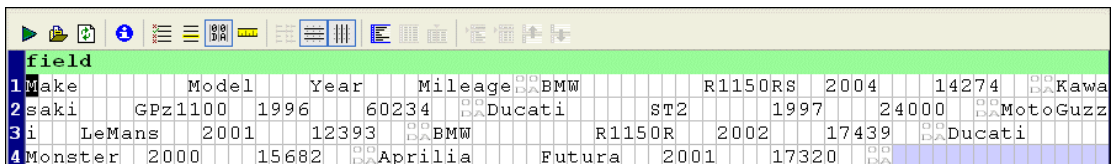
**Figure 178. Line-Oriented Region**

There are three 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)

### Example

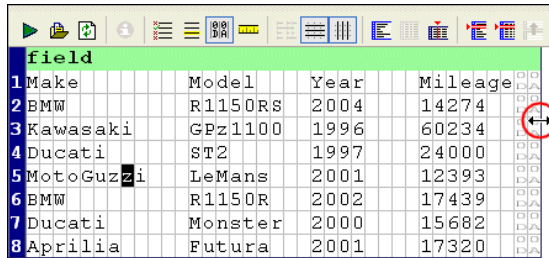
After converting the line-oriented region shown in [Figure 178](#) to fixed-width, it looks like this:



	field	field	field	field	field
1	Make		Model	Year	Mileage
2	saki	GPz1100	1996	60234	Ducati
3	i	LeMans	2001	12393	BMW
4	Monster	2000	15682	Aprilia	Futura

**Figure 179. Line-Oriented Region Converted to Fixed-Width**

Figure 178 shows the same fixed-width file after it has been resized by dragging the document pane.



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 180. 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 Convert to XML 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 257
- “Joining Regions” on page 258

### Defining a Region

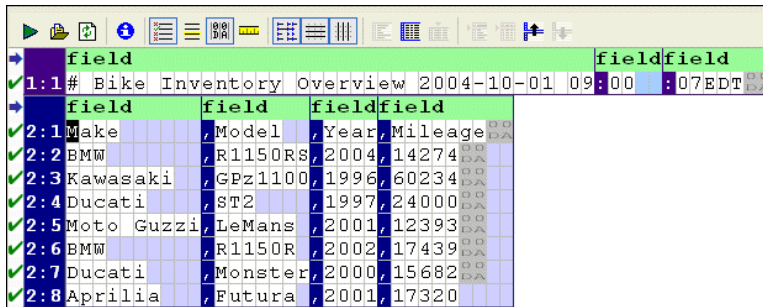
When you define a region in an input file, Stylus Studio splits the region at the current cursor location. The new region starts with the character on which the cursor resided when the region was defined, but it can be of either type – fixed-width or line-oriented – regardless of the type of the original 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 Convert to XML Editor rennumbers 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 181.





field				fieldfield			
1:1	#	Bike Inventory Overview	2004-10-01	09:00	:07EDT		
field				fieldfield			
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 181. 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 **ConvertToXML** menu and the shortcut menu in the Convert to XML Editor.

Stylus Studio defines the new region and rennumbers 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 20. 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 **ConvertToXML** menu and the shortcut menu in the Convert to XML 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. Unknown 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

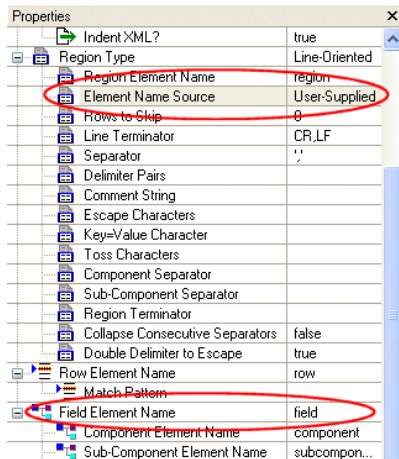
This section describes some of the features you can use to work with input file fields. It covers the following topics:

- “[Naming Fields](#)” on page 260
- “[Defining Fields](#)” on page 263
- “[Component and Sub-Component Fields](#)” on page 266

### 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 182](#).



**Figure 182. 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 Convert to XML 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:

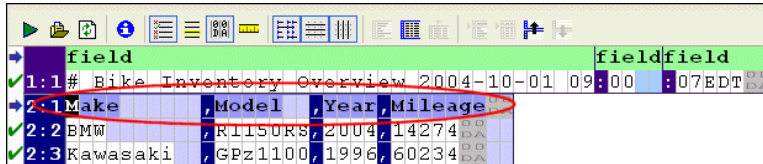
```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <row>
    <Make>BMW</Make>
    <Model>R1150RS</Model>
    <Year>2004</Year>
  </row>
  <row>
    <Make>MZ</Make>
    <Model>Scorpion</Model>
    <Year>1995</Year>
  </row>
  <row>
    <Make>Ducati</Make>
    <Model>ST2</Model>
    <Year>1997</Year>
  </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 183. 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

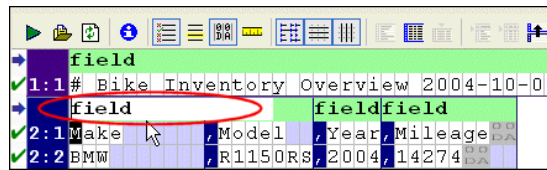
◆ **To provide user-supplied field names:**

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 184. You Can Edit the Field Name Directly in the Document Pane**

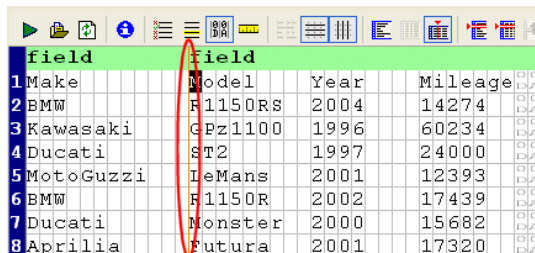
2. Type a new value for **field** and press Enter.

◆ **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 185](#). Once you have defined a field, you can change its size by simply dragging it to any column in the grid.



**Figure 185. 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 adapter. The input file shown in [Figure 185](#), 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.

## Converting Non-XML Files to XML

---

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>
  ...
```


If you specify fields for Model, Year, and Mileage, the XML output by the adapter 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 260.

◆ **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 **ConvertToXML** menu and the shortcut menu in the Convert to XML 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

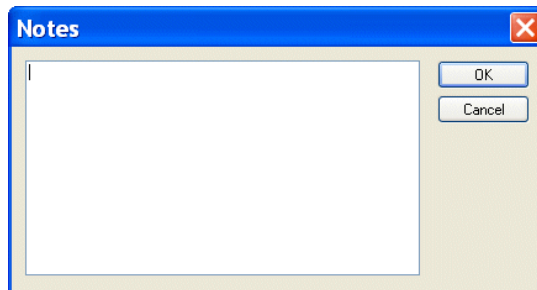
Stylus Studio allows you to create notes on individual fields. These notes are for reference purposes only; they are not output in the XML.

◆ **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 270 for more information on this topic.

The **Notes** dialog box appears.



**Figure 186. 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 Convert to XML 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 Convert to XML. Typically, the headers of these files contain information about the character used to specify component fields. Convert to XML uses this information to set the default value for the **Field Component Separator** property and render XML output accordingly.

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 Convert to XML 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

Convert to XML provides several ways for you to control the XML output created by the Convert to XML adapter. 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 Convert to XML Editor.

This section describes the properties used to control some of the most common output operations. See [“User-Defined Adapter Properties Reference”](#) on page 298 for detailed information on all properties.

This section covers the following topics:

- [“Specifying Element Names”](#) on page 268
- [“Specifying Format”](#) on page 269
- [“Omitting Regions and Fields, and Rows”](#) on page 269
- [“Pattern Matching”](#) on page 270
- [“Using Lookup Lists”](#) on page 275
- [“Using Key=Value Characters”](#) on page 277

### Specifying Element Names

You can specify names for the following in an XML document output by the adapter:

- 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 **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 the adapter.

- **Indenting** – By default, Stylus Studio indents the XML generated by the adapter. 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 `Unknown` 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 270 for more information.

## Pattern Matching

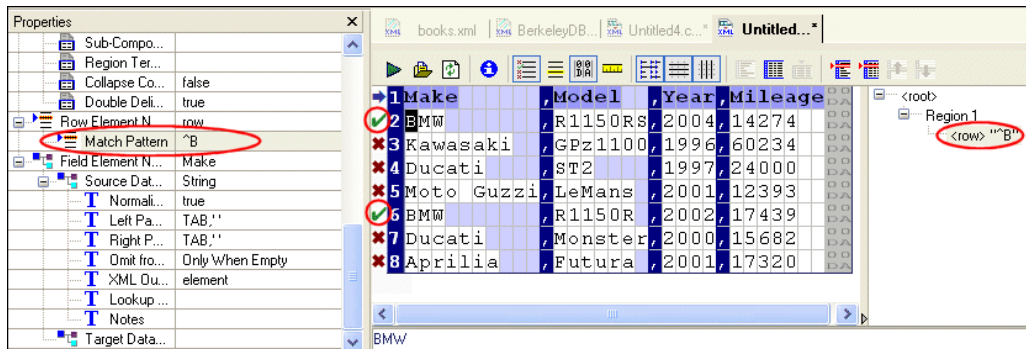
You can use regular expressions to specify match patterns for the rows in the input file. Only those rows in the input file that match the pattern you specify are output to XML when the file is converted. The simplest way to define a match pattern is to use the **Match Pattern** property in the **Row 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 Convert to XML Editor as shown in Figure 187 – 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 244 for more information about this feature.)



**Figure 187. 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 this adapter at this point, uses the default row element name (`row`) and the value of the expression.

Since the match pattern selects only those rows that begin with the letter B, the adapter 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>
```

See “[Working with Nodes](#)” on page 272 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.

<i>Expression</i>	<i>Matches</i>
<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”

To learn more about regular expression syntax, visit <http://www.boost.org/libs/regex/doc/syntax.html>.

## 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 188). 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 188, 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).

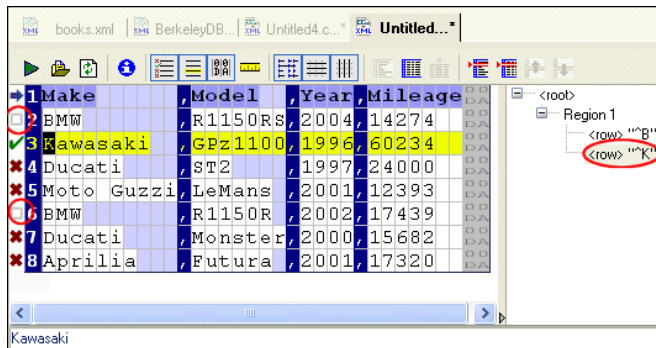


Figure 188. Gray Squares Identify Rows That Match Other Patterns

## 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 189. 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 **ConvertToXML > 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 **ConvertToXML > 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 **ConvertToXML > 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 adapter you are defining.

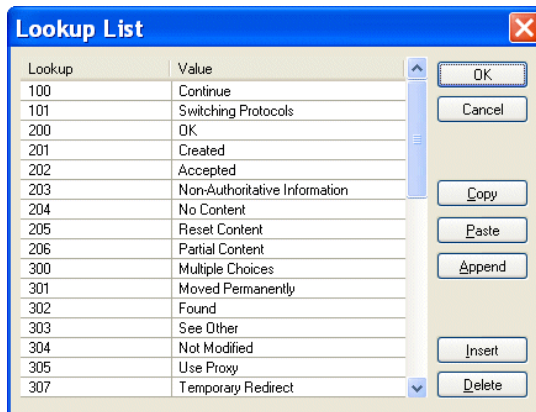
◆ **To remove a node:**

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 **ConvertToXML > 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 190](#) shows an example of a lookup list that has been defined for a Status field:



**Figure 190. 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:

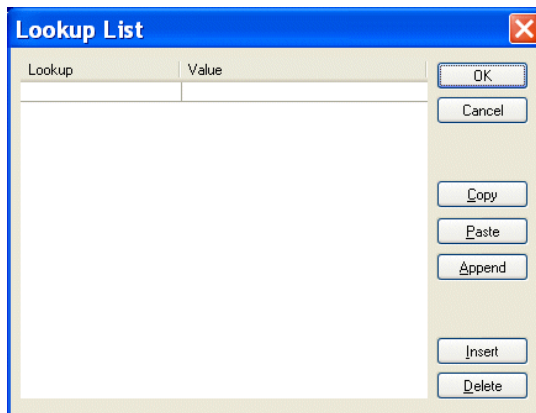
```
Make,Model,Year,Mileage
BMW,R1150RS,2004,14274
Kawasaki,GPz1100,1996,60234
Ducati,ST2,1997,24000
Moto Guzzi,LeMans,2001,12393
BMW,R1150R,2002,17439
Ducati,Monster,2000,15682
Aprilia,Futura,2001,17320
```

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 191. 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 21. Lookup List Dialog Box Buttons**

<i>Button</i>	<i>Function</i>
<b>OK</b>	Commits the lookup list to the adapter.
<b>Cancel</b>	Closes the <b>Lookup List</b> dialog box without committing any changes.
<b>Copy</b>	Copies the lookup list.
<b>Paste</b>	Pastes comma- and tab-separated text into the lookup list. Replaces existing content, regardless of which row you have selected
<b>Append</b>	Adds comma- and tab-separated text to the end of the lookup list. Existing lookup list content is preserved.
<b>Insert</b>	Adds a new row to the lookup list.
<b>Delete</b>	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
```

## Converting Non-XML Files to XML

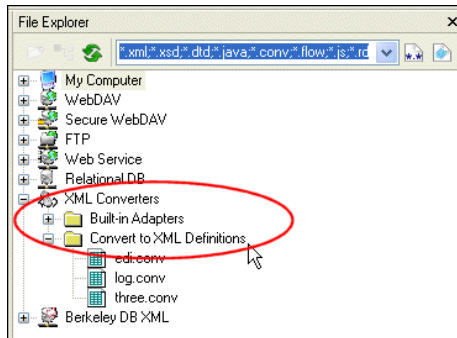
---

If you set the **Key=Value Character** property to =, Stylus Studio creates the following XML document when you preview the adapter:

```
<?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 an Adapter

This procedure describes how to create and save a Convert to XML adapter (.conv). All adapters are saved to the **Convert to XML Definitions** folder in the **XML Converters** file system by default, as shown in [Figure 192](#).



**Figure 192. User-Defined Adapters Saved to XML Converters File System**

## Specifying File Settings

When you create a new adapter, Stylus Studio prompts you to specify the file you want to convert. It also allows you to specify the file's

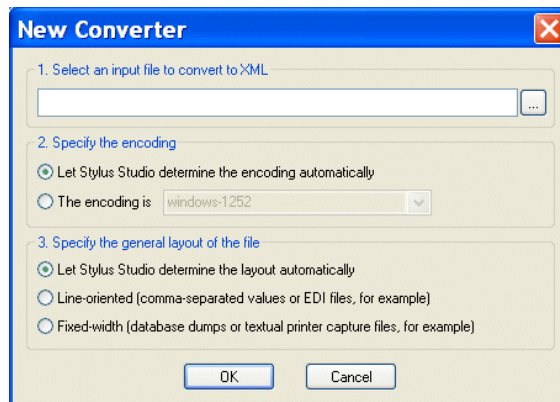
- Encoding (Windows-1252 or ANSI, for example)
- Layout (line-oriented or fixed-width, for example)

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 an Adapter

### ◆ To create an adapter (.conv file):

1. Select **File > New > Convert to XML** from the Stylus Studio menu. The **New Converter** dialog box appears.




**Figure 193. New Converter Dialog Box**

2. Click the browse button (  ) and select the file you want to use to configure the conversion 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.

3. Click the **Open** button.  
Stylus Studio displays the file URL in the first field of the **New Converter** dialog box.
4. Optionally, change the default file encoding and layout settings.
5. Click **OK**.  
Stylus Studio displays the file you selected in the Convert to XML Editor.
6. Examine the file and its properties as it was read into the Convert to XML Editor by Stylus Studio.
7. Modify the file layout, its regions, and its fields as needed. See [“Working with Regions”](#) on page 253 and [“Working with Fields”](#) on page 259 if you need help with this step.
8. Modify the properties that govern XML output as needed. See [“Controlling XML Output”](#) on page 267 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 adapter 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 reloading it.

## Using Adapters in Stylus Studio

You can use adapters 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 an adapter, the adapter engine converts that file to XML on-the-fly, using the settings defined in the adapter you select.

You can also use the Stylus Studio File System Java API to invoke an adapter programmatically to convert any file using the adapter you specify. See [“Invoking an Adapter Programmatically”](#) on page 287 for more information on this topic.

## Built-In Adapters

In addition to the adapters you create using the Convert to XML module (.conv files), Stylus Studio comes bundled with numerous predefined adapters to assist you in converting files to XML documents. They can be used to convert many common file formats, such as EDI, CSV, dBase, RTF, and Windows .ini files.

The process for using user-defined and built-in adapters is essentially the same, as described in the following section, “[How to Open a File Using an Adapter](#)” on page 281.

## How to Open a File Using an Adapter

You can use either of the following procedures to open a file using an adapter in Stylus Studio.

**Tip** Using the File Explorer can be easier – you can convert files using simple drag-and-drop.

### Using the File Explorer

**Note** Predefined adapters are displayed in the **Built-in Adapters** folder in the File Explorer.

◆ **To open a file using an adapter from the File Explorer:**

1. Navigate to the directory that contains the adapter you want to use to open the file.
2. Navigate to the directory that contains the file you want to convert.
3. Drag the file ([step 2](#)) and drop it on the adapter ([step 3](#)).

The file is converted to XML and opened as a new document.

### Using the Open Dialog Box

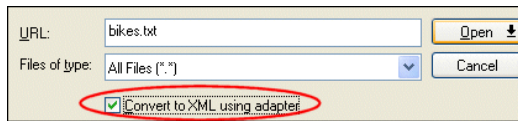
◆ **To open a file using an adapter from the Open dialog box:**

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 adapter and select the file.

## Converting Non-XML Files to XML

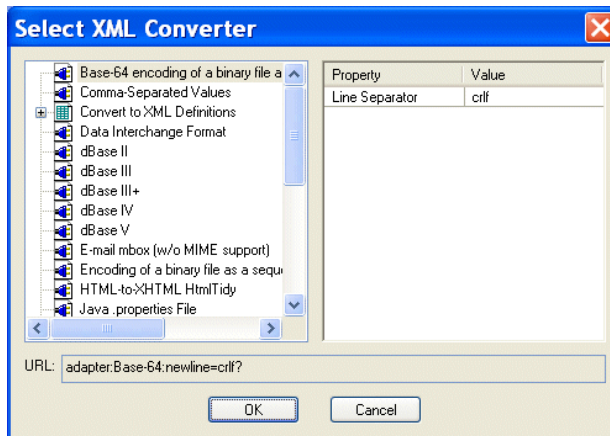
---

3. Check the **Convert to XML using adapter** check box and click the **Open** button.



**Figure 194. Check Box to Open Files Using Adapter**

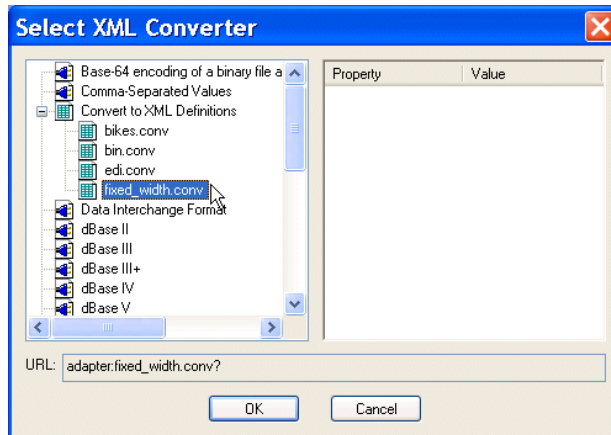
Stylus Studio displays the **Select XML Converter** dialog box.



**Figure 195. Select XML Converter Dialog Box**



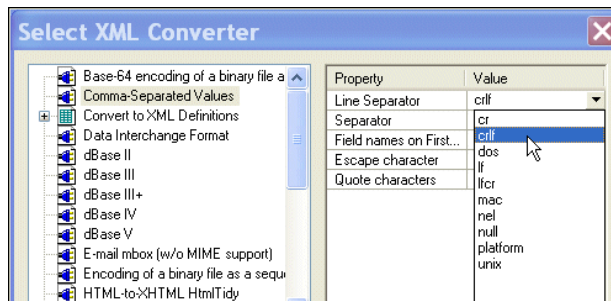
- If you are using a user-defined adapter, expand the **Convert to XML Definitions** tree and select the adapter you want to use to convert the file you selected in [step 2](#) to XML and go to [step 6](#).



**Figure 196. Selecting a User-Defined Converter**

If you are using a built-in adapter, select the adapter you want to use and go to [step 5](#).

- Specify the values you want to use for translation purposes. Many fields have a drop-down list that displays valid values, like the **Line Separator** property for the Comma-Separated Values converter shown here:



**Figure 197. Built-In Converters Display Valid Properties**

- 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 an adapter.

## More About Converting EDI

You can convert EDI to XML in Stylus Studio using the

- Convert to XML module to define the adapter manually
- Built-in Electronic Data Interchange (EDI) adapter to convert the files automatically

This section describes more about the level of support for converting EDI to XML (and vice versa) in Stylus Studio.

### Using Convert to XML for Converting EDI

As described earlier in this chapter (see [The Convert to XML Editor](#) on page 240), Stylus Studio's Convert to XML module has a user interface that provides you with complete control over how files, including EDI, are converted to XML. In the case of EDI files, EDIFACT, X12, and HL7 delimiter rules are handled automatically, and you can easily manage non-standard segments and extensions to other existing standards. The Convert to XML module also allows you to customize names for the fields in the output document and use regular expressions to filter the document's content.

The Convert to XML module is a useful way to jump-start EDI-to-XML conversion. If your needs include bi-directional conversion (XML-to-EDI as well as EDI-to-XML), consider using Stylus Studio's built-in Electronic Data Interchange (EDI) adapter.

### Using the Built-In EDI Adapter

The Electronic Data Interchange (EDI) adapter allows you to convert EDI to XML, and vice versa. It handles most versions of EDIFACT standard vocabularies automatically, and it optionally performs validation of content, structure, and code list values. The

following table summarizes the adapter settings that affect how EDI is converted to XML. Note that these settings are used for EDI-to-XML conversions only.

**Table 22. Built-In EDI Adapter Properties**

<i>Property</i>	<i>Description</i>
Line separator	The line separator character in the EDI file. Valid values are displayed in a drop-down list.
Enable validation	Whether or not you want to enable validation. Setting this property to No turns off all validation. Only errors that keep the parser from determining the element type that are necessary for decoding are reported in this case. By default, validation is on.
Comment code list table data	Whether or not you want Stylus Studio to emit XML comments showing the decoded value for each item that has a value in a code list. This property is set to Yes by default.
Comment element types	Whether or not you want Stylus Studio to emit XML comments detailing the type of each element in the output. This property is set to Yes by default.
Strict validation on value lengths	Whether or not you want Stylus Studio to validate the exact or maximum length of elements as declared in the specification. This property is set to No by default.
Strict segment-ordering checking	Whether or not you want Stylus Studio to slightly relax the enforcement of segments to be in the proper sequence. If this property is set to Yes, conversion still proceeds, but this can cause cascade errors if too many segments are omitted – that is, it is possible for the decoder to lose its place since it has no reference points to resynchronize its decoding with the specification. This property is set to No by default.

**Table 22. Built-In EDI Adapter Properties**

<i>Property</i>	<i>Description</i>
Force error if value not in code list	Whether or not you want Stylus Studio to abort the conversion if a value is supposed to be from a code list but it is not found in the code list. Though this property is set to Yes by default, you might want to consider setting it to No to support extensions to the code list tables provided by the producer.
Strict datatype content checking	Whether or not you want Stylus Studio to validate the contents of each value, to make sure there are no digits in alpha-only elements, or no non-digits in number-only elements. This property is set to Yes by default.
Treat all segments as optional	Some segments are mandatory, others are optional. For producers that only generate a subset of the required segments, this property (if set to Yes) turns off error checking if an expected segment is omitted. This is similar behavior to the segment order checking, but heuristics are used in an effort to determine what the producer has generated. This property is set to No by default.

### Where to Find It

As with other adapters, you can access the Electronic Data Interchange (EDI) adapter in the

- **Built-in Adapters** folder of the **XML Converters** file system displayed in the Stylus Studio **File Explorer** window
- **Select XML Converter** dialog box, which appears when you open a file using the **Convert to XML using adapter** check box on the Open dialog box

If you want to modify the default property settings, you must open the adapter from the **Select XML Converter** dialog box. When accessed through the **File Explorer** window, the adapter uses whatever property settings have been set through the **Select XML Converter** dialog box.

## Validating XML from/to EDI

If you plan to use the Electronic Data Interchange (EDI) adapter to convert XML to EDI, you can use the EDIFACT to XML Schema document wizard to create an XML Schema based on a specific EDIFACT message type. This allows you to ensure that the EDI created by the Electronic Data Interchange (EDI) adapter conforms to the EDIFACT message type standard.

You can also XML Schema created by the EDIFACT to XML Schema document wizard 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 EDIFACT Messages](#) on page 506 for more information on using this document wizard.

## Invoking an Adapter Programmatically



The File System Java API is available only in Stylus Studio XML Enterprise Edition and Stylus Studio XML Professional Edition.

You can use the Stylus Studio File System Java API to invoke an adapter programmatically. This section provides background information on using adapter URLs in Stylus Studio and in your applications, and it describes `demo.bat`, a simple Java program installed with Stylus Studio.

This section covers the following topics:

- [“Adapter URLs”](#) on page 287
- [“The StylusFile Object”](#) on page 290
- [“Constructing Your Own Adapter URL”](#) on page 291
- [“Example – demo.bat”](#) on page 293
- [“More About the Stylus Studio File System Java API”](#) on page 298

## Adapter URLs

Stylus Studio uses URLs extensively to reach a variety of data sources. For example, if you use the built-in CSV adapter to open the `three.txt` file in the `\examples\Adapters` directory as an XML document, Stylus Studio builds the following URL:

## Converting Non-XML Files to XML

---

adapter:CSV:newline=crLf:sep=,:first=no:escape=\:quotes=""?file:///c:/Program Files/Stylus Studio XML Professional Edition/examples/Adapters/three.txt

The instructions to the adapter engine from this instance of the adapter URL are described in [Table 23](#).

**Table 23. Parts of an Adapter URL**

<i>Instruction</i>	<i>Adapter URL String</i>
Use the Comma-Separated Values adapter	adapter:csv
The line separator in the source file is a carriage return/line feed	newline=crLf
The column separator in the source file is a comma	sep=,
The values in the first row of the source file should be used to supply field names	first=yes
The escape character in the source file is a slash \	escape=\
The quote characters in the source file are " and '	quotes=""
The source file is three.txt	file:///c:/Program Files/Stylus Studio XML Professional Edition/examples/Adapters/three.txt

While the basic format of the adapter URL is the same from one adapter to another, there are differences. For example:

- Built-in adapters have different properties. You should always use Stylus Studio to build your adapter URLs to be sure that it uses valid properties and values. See [“Constructing Your Own Adapter URL”](#) on page 291 for more information.
- The adapter scheme can also be used to reference a user-defined adapter (a .conv file). In this case, the adapter URL specifies only the location of the .conv file; the adapter file itself contains information about its property settings. An adapter URL that references a user-defined adapter might look like this:

```
adapter:///myAdapter.conv?file:inventory.txt
```

This adapter uses `myAdapter.conv` to convert the file `inventory.txt` to some format (specified in the adapter file when you built it).

- Property names in adapter URLs use an internal (generally, abbreviated) format, different from what is displayed in properties list in the **Select XML Converter** dialog box. Again, to avoid errors in your applications, use Stylus Studio to build your adapter URLs.

### Where Adapter URLs are Displayed in Stylus Studio

Adapter URLs are displayed

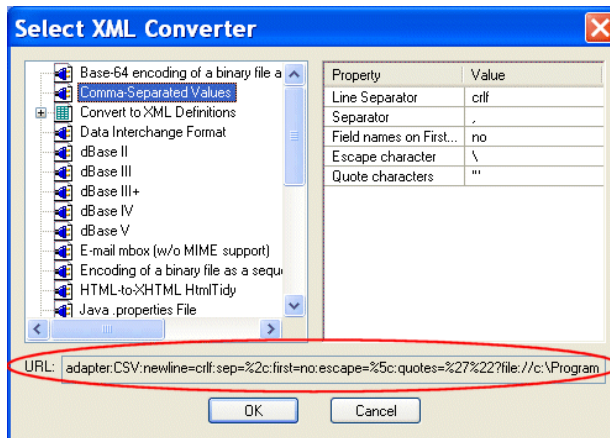
- In the **Project** window (select **Show Full URL** from the **Project** window shortcut menu)



**Figure 198. Adapter URL Displayed in Project Window**

## Converting Non-XML Files to XML

- In the **URL** field of the **Select XML Converter** dialog box, as shown in [Figure 199](#).



**Figure 199. Adapter URL Displayed in the Select XML Converter Dialog Box**

**Note** Adapter URLs in the **URL** field of the **Select XML Converter** dialog box are displayed using escape characters.

You can use either as sources for the adapter URL strings in your Java applications. See [“Constructing Your Own Adapter URL”](#) on page 291 for more information.

## The StylusFile Object

In Java, we write the following to retrieve the content of a file, `myfile.txt`, that resides on an FTP server:

```
java.net.URL myUrl = new java.net.URL("ftp://myserver/myfile.txt");
java.io.InputStream inputStream = myUrl.openStream();
```

The content of `myfile.txt` is placed in an instance of `java.net.URL` called `myUrl`.

The `StylusFile` object in the Stylus Studio File System Java API is similar to `java.net.URL`. In Stylus Studio, you use the `createStylusFile()` method in the `StylusFileFactory` class to create an instance of a `StylusFile` object, the contents of which is specified by an adapter URL. Consider the following simple Java application,



which uses the URL composed by Stylus Studio that was introduced in “[Adapter URLs](#)” on page 287.

```
StylusFile myStylusFile =
StylusFileFactory.getFactory().createStylusFile("adapter:CSV:newline=crlf:
sep=, :first=yes:escape=/:quotes='\"?file:///c:/Program Files/Stylus Studio
XML Professional Edition/examples/Adapters/three.txt");

java.io.InputStream inputStream = myStylusFile.getInputStream();
```

In this example, the `InputStream` object holds the conversion result supplied by the adapter URL (that is, the conversion of `three.txt` using the CSV adapter).

See “[Example – demo.bat](#)” on page 293 for a description of the sample Java application installed with Stylus Studio, which illustrates different uses of the adapter URL.

## Constructing Your Own Adapter URL

Generally speaking, you should use Stylus Studio to construct adapter URLs, which you can then copy, as strings, into a Java application. Adapter URLs can be complex – properties and their values vary from one adapter to another, for example – so using Stylus Studio to construct them helps reduce errors in your applications.

**Tip** Adapter URLs in the **Select XML Converter** dialog box are already escaped and can be used as-is. Adapter URLs taken from the **Project** window must be escaped manually when you paste them into your application.

## Using the URL in the Select XML Converter Dialog Box

### ◆ To construct an adapter URL using the URL in the Select XML Converter dialog box:

1. Use the adapter to open a file as an XML document in Stylus Studio. This can be a user-defined or built-in adapter, as appropriate. See “[How to Open a File Using an Adapter](#)” on page 281 if you need help with this step.
2. Before clicking **OK** to complete the conversion, copy the adapter URL in the **URL** field of the **Select XML Converter** dialog box (see [Figure 199](#)).
3. Click **OK** to complete the conversion.
4. Paste the adapter URL into your Java program.

### Using the URL in the Properties Window

◆ **To construct an adapter URL using the URL in the Properties window:**

1. Use the adapter to open a file as an XML document in Stylus Studio. This can be a user-defined or built-in adapter, as appropriate. See “[How to Open a File Using an Adapter](#)” on page 281 if you need help with this step.

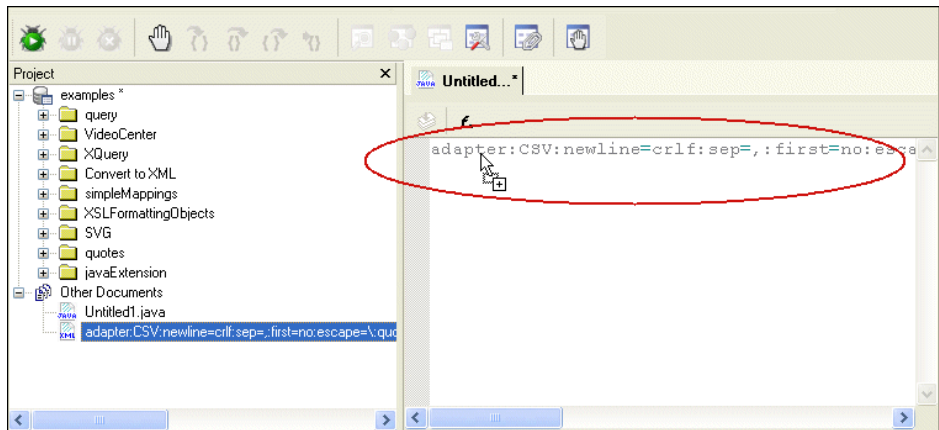
**Tip** New documents are placed in the **Other Documents** folder in the **Project** window by default.

2. Open a new document in any Stylus Studio text editor (for example, **File > New > Java: Text Editor**).

The purpose of this step is to provide an editor into which you can drag-and-drop the adapter URL for the document you created in [step 1](#).

3. Drag the document you created in [step 1](#) from the **Project** window and drop it into the text editor associated with the document you created in [step 2](#).

The complete URL appears in the text editor.



**Figure 200. Copying a URL to a Text Editor**

4. Copy the complete adapter URL.
5. Paste the adapter URL into your Java program.
6. 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).

## Example – demo.bat

The following example shows an implementation of a simple application built using the Stylus Studio File System Java API. The application, `demo.bat`, shows three uses of invoking adapter URLs to convert files using built-in and user-defined adapters.

### Demonstration Files

The files required to run this demonstration are installed in the `\examples\Adapters` directory where you installed Stylus Studio. They are summarized in [Table 24](#).

**Table 24. File System Java API Demonstration Files**

<i>Class</i>	<i>Description</i>
<code>demo.bat</code>	The demonstration driver batch file.
<code>demo.class</code>	The compiled class file.
<code>demo.java</code>	The source for the demonstration; this file contains the usage comments.
<code>one.csv</code>	The input file for the first demonstration run by <code>demo.bat</code> .
<code>two.xml</code>	The input file for the second demonstration run by <code>demo.bat</code> .
<code>three.conv</code>	The adapter definition for the third demonstration run by <code>demo.bat</code> . (This file should have also been installed in your Stylus Studio <code>\Adapter</code> directory.)
<code>three.txt</code>	The input file for the third demonstration run by <code>demo.bat</code> .
<code>adaptermap.properties</code>	Normally, Stylus Studio builds a map of all of the internally-supplied adapters and user-added adapters, and it places this map in a file called <code>adaptermap.properties</code> . For this demonstration, a small copy of the <code>adaptermap.properties</code> file is included in the <code>\examples\Adapters</code> directory.

### demo.java

Here is the demo.java file called by demo.bat.

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.IOException;

import com.exln.stylus.io.StylusFile;
import com.exln.stylus.io.StylusFileFactory;

public class demo {
    public static void main(String args[]) {
        StylusFileFactory sff = StylusFileFactory.getFactory();

        try {
            StylusFile in_1 =
sff.createStylusFile("adapter:///CSV:sep=, :first=yes?file:one.csv");
            File out_1 = new File("one.xml");
            InputStream is = in_1.getInputStream();
            OutputStream os = new FileOutputStream(out_1);
            copy(is, os);
            in_1.close();
            System.out.println("test 1 succeeded: one.csv -> one.xml");
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }

        try {
            File in_2 = new File("two.xml");
            StylusFile out_2 =
sff.createStylusFile("adapter:///CSV:sep=, :first=yes?file:two.csv");
            InputStream is = new FileInputStream(in_2);
            OutputStream os = out_2.getOutputStream();
            copy(is, os);
            out_2.close();
            System.out.println("test 2 succeeded: two.xml -> two.csv");
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
}
```

```

        try {
            StylusFile in_3 =
sff.createStylusFile("adapter:///three.conv?file:three.txt");
            File out_3 = new File("three.xml");
            InputStream is = in_3.getInputStream();
            OutputStream os = new FileOutputStream(out_3);
            copy(is, os);
            in_3.close();
            System.out.println("test 3 succeeded: three.txt -> three.xml");
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
        static void copy(InputStream is, OutputStream os) throws IOException {
            byte buffer[] = new byte[8192];
            int bytesRead;
            while ((bytesRead = is.read(buffer)) != -1) {
                os.write(buffer, 0, bytesRead);
            }
            os.flush();
            os.close();
        }
    }
}

```

### Required classes

The file starts with the Java and Stylus Studio File System Java API classes and interfaces required by `demo.java`. The `StylusFile` interface is used to define an abstract representation of a file in a custom file system; the `StylusFileFactory` class enables the `demo.bat` application to read and write files.

```

import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.IOException;

import com.exln.stylus.io.StylusFile;
import com.exln.stylus.io.StylusFileFactory;

```

### Setting the installation directory

User-defined adapters (`.conv` files) are saved to the **Convert to XML Definitions** folder in the **XML Converters** file system. If your application uses user-defined adapters, such as the `three.conv` file used by the `demo.bat`, you need to identify the Stylus Studio

installation directory so that the adapter can be located. You can do this using `System.setProperty`, as shown in the following lines of code.

```
public class demo {
    public static void main(String args[]) {
        // System.setProperty("com.stylusstudio.rootdir", "C:\\Program
        Files\\Stylus Studio XML Professional Edition");
    }
}
```

These lines are commented out in `demo.java` because you can also specify the Stylus Studio installation directory using the `-D` argument at the command line, as shown in `demo.bat`: `"-Dcom.stylusstudio.rootdir=%STYLUS%"`).

### Creating an instance of StylusFile

Next, the `StylusFileFactory` class is called to create a new instance of the `StylusFile`, object, named `sff` in this example.

```
StylusFileFactory sff = StylusFileFactory.getFactory();
```

### The copy () method

The `copy()` method is used to create new files by saving the file specified in the `InputStream` (`is`) to the file specified in the `OutputStream` (`os`).

The `copy()` method is defined in `demo.java` as follows:

```
static void copy(InputStream is, OutputStream os) throws IOException {
    byte buffer[] = new byte[8192];
    int bytesRead;
    while ((bytesRead = is.read(buffer)) != -1) {
        os.write(buffer, 0, bytesRead);
    }
    os.flush();
    os.close();
}
```

Whether your files are saved as XML or non-XML depends on how the `InputStream` and the `OutputStream` in your applications are defined. You can define either stream using an adapter URL. When the

- `InputStream` is specified using the adapter URL, a non-XML file is converted to XML using the adapter specified in the adapter URL.
- `OutputStream` is specified using the adapter URL, an XML document is converted to a non-XML file, again, using the adapter specified in the adapter URL.

## Converting a file to XML

Once the new instance of `StylusFile` has been created, the `demo.bat` application can execute its first exception block. The adapter URL specified in this exception block uses the built-in Comma-Separated Values adapter (`adapter:///CSV:`) to convert a CSV file named `one.csv` into an XML document named `one.xml`.

```
try {
    StylusFile in_1 =
sff.createStylusFile("adapter:///CSV:sep=,:first=yes?file:one.csv");
    File out_1 = new File("one.xml");
    InputStream is = in_1.getInputStream();
    OutputStream os = new FileOutputStream(out_1);
    copy(is, os);
    in_1.close();
    System.out.println("test 1 succeeded: one.csv -> one.xml");
} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

In this block of code, the adapter URL is used to specify the `InputStream`.

## Converting an XML document to another format

This exception block specifies the `OutputStream` using the adapter URL to convert an XML document named `two.xml` to a CSV file named `two.csv`.

```
try {
    File in_2 = new File("two.xml");
    StylusFile out_2 =
sff.createStylusFile("adapter:///CSV:sep=,:first=yes?file:two.csv");
    InputStream is = new FileInputStream(in_2);
    OutputStream os = out_2.getOutputStream();
    copy(is, os);
    out_2.close();
    System.out.println("test 2 succeeded: two.xml -> two.csv");
} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

### Using an adapter URL with a user-defined adapter

The final exception block uses a user-defined adapter built using the Convert to XML module (`three.conv`) to convert a fixed-width file name `three.txt` to an XML document named `three.xml`.

```
try {
    StylusFile in_3 =
sff.createStylusFile("adapter:///three.conv?file:three.txt");
    File out_3 = new File("three.xml");
    InputStream is = in_3.getInputStream();
    OutputStream os = new FileOutputStream(out_3);
    copy(is, os);
    in_3.close();
    System.out.println("test 3 succeeded: three.txt -> three.xml");
} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

As with the first exception block, this block uses the adapter URL to specify the `InputStream`.

## More About the Stylus Studio File System Java API

You can learn about other uses for the Stylus Studio File System Java API in “[Stylus Studio File System Java API](#)” on page 811, located in “[Extending Stylus Studio](#)” on page 795.

### Javadoc

Javadoc for the Stylus Studio File System Java API is installed with Stylus Studio. It is available in the Stylus Studio `/doc/Javadoc` directory. Open [index.html](#) to get started. The Javadoc is also available on the Stylus Studio Web site, <http://www.StylusStudio.com>.

## User-Defined Adapter Properties Reference

This section provides reference information for input file properties displayed in the **Properties** window. It covers the following topics:

- “[Input File Properties](#)” on page 299
- “[XML Output File Properties](#)” on page 300
- “[Region Properties](#)” on page 301
- “[Row Properties](#)” on page 304



- [“Field Properties”](#) on page 305
- [“Type-Specific Properties”](#) on page 307
- [“Specifying Control Characters”](#) on page 311

## Input File Properties

**Table 25. 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
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 an adapter. The finished adapter, 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
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 25. Input File Properties**

<b><i>Property</i></b>	<b><i>Description</i></b>	<b><i>Editable</i></b>	<b><i>Affects XML</i></b>
Output Encoding	The Encoding you want to use for the XML output by the adapter. The default is RAW.	Yes	Yes
Indent XML?	Whether or not you want to indent the XML output.	Yes	Yes

## XML Output File Properties

**Table 26. XML Output File Properties**

<b><i>Property</i></b>	<b><i>Description</i></b>	<b><i>Editable</i></b>	<b><i>Affects XML</i></b>
XML Output URL	The URL to which you want the XML output when the adapter is run. Optional. If no value is specified, stdout is used.	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
Namespace	The namespace you want to use for the XML document output by the adapter.	Yes	Yes
Output Encoding	The Encoding you want to use for the XML output by the adapter. The default is RAW.	Yes	Yes
DOCTYPE System ID	The URL of the System DTD you want to associate with the XML document output by the adapter.	Yes	Yes

Table 26. XML Output File Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
DOCTYPE Public ID	The Public ID of the DTD you want to associate with the XML document output by the adapter.	Yes	Yes
XML Schema Document	The URL of the XML Schema document you want to associate with the XML document output by the adapter.	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 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 27. 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

**Table 27. Region Properties**

<b>Property</b>	<b>Description</b>	<b>Editable</b>	<b>Affects XML</b>
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 <sup>+</sup>	The number of characters in the current row.	No	No
Line Terminator*	The type of line terminator character detected by Stylus Studio.	Yes	No
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 Convert to XML adapter – if the beginning of a row matches the string in this field, the adapter interprets the row as a comment and does not output it in the XML.	Yes	Yes

**Table 27. Region Properties**

<b>Property</b>	<b>Description</b>	<b>Editable</b>	<b>Affects XML</b>
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
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

**Table 27. Region Properties**

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
Collapse Consecutive Field Separators*	Whether or not multiple consecutive field separators should be treated as one. For example, if this property is set to <b>Yes</b> , 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 <b>Yes</b> , "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 Properties

**Table 28. 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

Table 28. Row Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
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 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. Field Properties

<i>Property</i>	<i>Description</i>	<i>Editable</i>	<i>Affects XML</i>
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

**Table 29. Field Properties**

<b>Property</b>	<b>Description</b>	<b>Editable</b>	<b>Affects XML</b>
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 <a href="#">“Type-Specific Properties”</a> on page 307 for information on properties that are associated with specific data types.	No	Yes
Target Data Type	<i>Not currently used.</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.



## Type-Specific Properties

This section describes the properties that are specific to a given data type.

**Table 30. Type-Specific Properties**

<i>Property</i>	<i>Applies to Data Type</i>	<i>Description</i>	<i>Affects XML</i>
Architecture	BCD	Specifies the architecture type for Binary Coded Decimal (BCD).	Yes
Base	Number	The numeric base of the number. Decimal (base 10) is assumed by default, but base 2 through base 36 are recognized.	Yes
'C' rules for octal and hex	Number	Determines how numeric values are treated. If this property is set to True and the value begins with 0x, the value is treated as hexadecimal (base 16). If the value begins with 0, it is treated as octal (base 8). Otherwise, it is treated as base 10.	Yes
Decimal	Number	The decimal separator character. Defaults to the standard platform symbol, which is typically a period or comma.	Yes
Endian	Double, Float, Integer, Long, Short	Determines whether the first byte in binary multi-byte values is the least significant (Little) or the most significant (Big). The default is Little, which is the native mode on Stylus Studio platforms.	Yes
<ul style="list-style-type: none"> <li>● False Output As</li> <li>● True Output As</li> <li>● Unknown Output As</li> </ul>	Boolean	Strings output based on matching strings specified in the False Value Match List and True Value Match List.	Yes

**Table 30. Type-Specific Properties**

<i>Property</i>	<i>Applies to Data Type</i>	<i>Description</i>	<i>Affects XML</i>
<ul style="list-style-type: none"> <li>● False Value Match List</li> <li>● True Value Match List</li> </ul>	Boolean	<p>Semicolon-separated lists of values. If the field value matches a string specified in the False Value Match List, the string specified in the False Output As property is output; likewise for values that match those specified in the True Value Match List.</p> <p>If there are no matches with either the false or true match lists, the value specified in the Unknown Output As property is output.</p>	Yes
Format	Date	<p>Specifies the order of the day, month, and year in date strings. Convert to XML can determine date format if names are used for months.</p> <p>Dates are recognized in many formats, with and without delimiting characters, including:</p> <p>MMDDYY            MMDDYYYY            DDMYY            DDMYYYY            YYMMDD            YYYYMMDD            MM-DD-YY            MM-DD-YYYY            DD-MM-YY            DD-MM-YYYY            YY-MM-DD            YYYY-MM-DD            YYJJJ            YYYYJJJ            YY-JJJ            YY-JJJJ</p>	Yes

Table 30. Type-Specific Properties

<b>Property</b>	<b>Applies to Data Type</b>	<b>Description</b>	<b>Affects XML</b>
<ul style="list-style-type: none"> <li>● Left Padding</li> <li>● Right Padding</li> </ul>	Boolean, Date, Number, String, Time	Allows you to specify characters you want removed from the left and right sides, respectively, of a field. Applies to characters within delimiting characters.	Yes
Lookup List	All	Allows you to specify a list of lookups and associated values. Values are substituted for the lookup when the input file is converted to XML.	Yes
Normalize White Space	String	Converts any tabs, linefeeds, or carriage returns to spaces. Also removes leading and trailing white space and collapses consecutive white space characters to a single space. The default is <code>False</code> .	Yes
Notes	All	Allows you to add internal comments for a field.	No
Omit from Output	All	Allows a field to be omitted from XML output based on its presence or value. Valid values include: <ul style="list-style-type: none"> <li>● Only When Empty. This is the default.</li> <li>● When Empty or Zero.</li> <li>● Always</li> <li>● Never</li> </ul>	Yes
Packed	BCD	For Binary Coded Decimal (BCD) fields, whether or not it is packed.	Yes

**Table 30. Type-Specific Properties**

<b>Property</b>	<b>Applies to Data Type</b>	<b>Description</b>	<b>Affects XML</b>
Scaling Factor( $10^n$ )	BCD, Byte, Comp3, Double, Float, Integer, Long, Number, Short, Zoned	Scales a number by moving the decimal point. Use a positive value to move the decimal point to the left; use a negative value to move the decimal point to the right. 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
Separator	Date, Time	Character used to separate date and time components. Date values commonly use spaces, slashes, dashes, and periods; time values commonly use colons, periods, and spaces. The platform setting is used by default.	Yes
Signed	Byte, Integer, Long, Short	Whether the numeric form is signed or unsigned.	Yes
Thousands Separator	Number	Character used to separate thousands. Typically a comma, though periods, spaces, and other characters can be used. Defaults to the platform symbol.	Yes

Table 30. Type-Specific Properties

<b>Property</b>	<b>Applies to Data Type</b>	<b>Description</b>	<b>Affects XML</b>
Use Currency Conventions	Number	Determines whether values are treated as positive or negative. If the value contains DR or DB (or dr or db), it is treated as positive. If it contains CR or cr, or is surrounded by parentheses, it is treated as a negative.	Yes
Window for Two-Digit Years	Date	Allows you to specify the century start-date for years that use a two-digit format. For example, if you enter 1950, years with a value from 50-99 will be output as 1950-1999; years with a value of 00-49 will be output as 2000-2049.	Yes
XML Output Form	All	Whether to output the current row value as a child element or attribute of the <row> element. The default is element. If this field is set to True, components and subcomponents within that value are not recognized, so the entire field is used as the attribute value, instead of being broken up.	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 31. 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 31. 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	VTS	
139	0x8B	PLD	

**Table 31. 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 discusses the following topics:

- [“Getting Started with XSLT”](#) on page 315
- [“Tutorial: Understanding How Templates Work”](#) on page 339
- [“Working with Stylesheets”](#) on page 351
- [“Creating Stylesheets That Generate HTML”](#) on page 369
- [“Specifying Extension Functions in Stylesheets”](#) on page 376
- [“Working with Templates”](#) on page 381
- [“Using an External XSLT Processor”](#) on page 386
- [“Validating Result Documents”](#) on page 389
- [“Post-processing Result Documents”](#) on page 391
- [“Generating Formatting Objects”](#) on page 392
- [“Generating Scalable Vector Graphics”](#) on page 398
- [“Generating Java Code for XSLT”](#) on page 399
- [“XSLT Instructions Quick Reference”](#) on page 405

### **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 316
- [“What Is a Stylesheet?”](#) on page 317

- [“What Is a Template?”](#) on page 320
- [“How the XSLT Processor Applies a Stylesheet”](#) on page 323
- [“Controlling the Contents of the Result Document”](#) on page 329
- [“Specifying XSLT Patterns and Expressions”](#) on page 331
- [“Frequently Asked Questions About XSLT”](#) on page 333
- [“Sources for Additional XSLT Information”](#) on page 334
- [“Benefits of Using Stylus Studio”](#) on page 335

## 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 613.
- 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 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 317
- [“About Stylesheet Contents”](#) on page 320

## 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

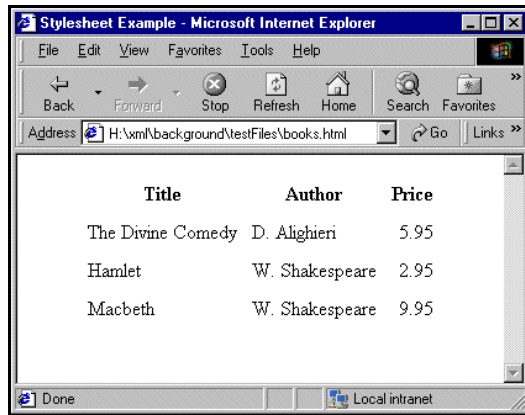
For example, suppose you have the following XML document:

Source  
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>
```

Resulting  
HTML file

You can use a stylesheet to transform this XML document into an HTML document that appears as follows in a Web browser:



**Figure 201. Example of Transformed XML**

The Web page in [Figure 201](#) 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).

Stylesheet  
used

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

Standard XML declaration. Following is a stylesheet that does this.

`xsl:stylesheet` is an XSLT instruction. It must be the root element in a stylesheet to be used with 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

```
<?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>
```

`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

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

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 format.

`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 432
  - “[xsl:output](#)” on page 425
  - “[xsl:template](#)” on page 432
  - “[xsl:apply-templates](#)” on page 406
  - “[xsl:sort](#)” on page 429
  - “[xsl:value-of](#)” on page 435

Each XSLT element is an instruction to the XSLT processor. For information about all XSLT instructions, see “[XSLT Instructions Quick Reference](#)” on page 405.

- Non-XSLT elements and attributes. In the previous stylesheet, these include the HTML elements that create the table.

The Root  
element

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 321
- [Determining Which Template to Instantiate](#) on page 322
- [How the select and match Attributes Are Different](#) on page 323

## Contents of a Template

The [stylesheet example](#) in “[Stylesheet used](#)” on page 318 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 “[Stylesheet used](#)” on page 318, 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 “[Stylesheet used](#)” on page 318 to present this process in more detail in the following topics:

- [Instantiating the First Template](#) on page 324
- [Selecting Source Nodes to Operate On](#) on page 325
- [Controlling the Order of Operation](#) on page 326
- [Omitting Source Data from the Result Document](#) on page 327
- [When More Than One Template Is a Match](#) on page 328
- [When No Templates Match](#) on page 328

### 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 429.

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 `xsl:apply-templates` instruction. When the XSLT processor executes the `xsl:apply-templates` instruction, it performs the following steps:

1. The processor evaluates the expression specified for the `xsl: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 `xsl: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 `xsl: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 `xsl:apply-templates` instruction. In this way, the XSLT processor can descend many levels to complete processing of the first selected node in the initial `xsl:apply-templates` instruction. The `xsl: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 324 contains the following `xsl:apply-templates` instruction:

```
<xsl: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 405.

## 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 [“Source XML document”](#) on page 317 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 432 and [“xsl:apply-templates”](#) on page 406.

### 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 383.

## 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 329
- [Creating New Nodes in the Result Document](#) on page 330
- [Controlling White Space in the Result](#) on page 330

### Specifying Result Formatting

In a stylesheet, you can specify that the XSLT processor should format the result as XML, HTML, or text. [Table 32](#) describes the XSLT processor output for each alternative:

**Table 32. 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 425 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 405:

- [“xsl:element”](#) on page 416
- [“xsl:attribute”](#) on page 407
- [“xsl:comment”](#) on page 412
- [“xsl:processing-instruction”](#) on page 428
- [“xsl:text”](#) on page 434

You can use the [xsl:value-of](#) on page 435 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 413.

### 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 613.

## 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 486.



XSLT Profiling is available only in Stylus Studio XML Enterprise Edition.

### 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 `<br/>`. How can I output just `<br>`?**

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 386.

*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 425.

## 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 335
- [Sophisticated Editing Environment](#) on page 336
- [XSLT and Java Debugging Features](#) on page 337
- [Integrated XML Parser/XSLT Processor](#) on page 339

### Structural Data View

Stylus Studio graphically displays the structure, or schema, of the XML data to which you want to apply a stylesheet.

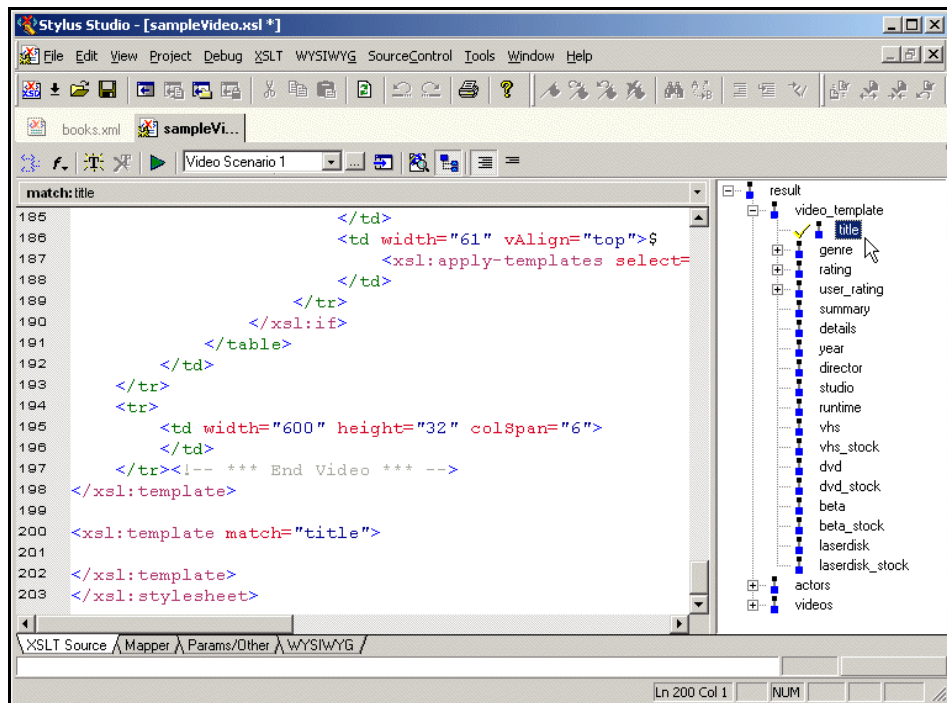
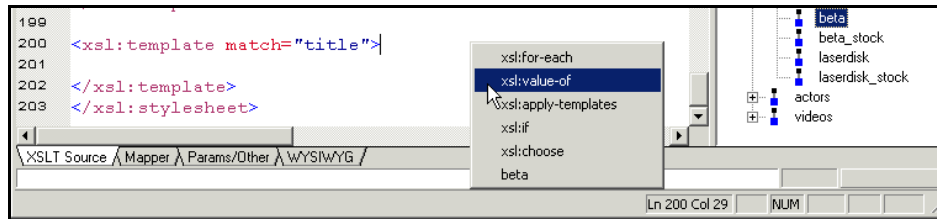


Figure 202. Tree View Lets You Easily Edit XSLT

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.



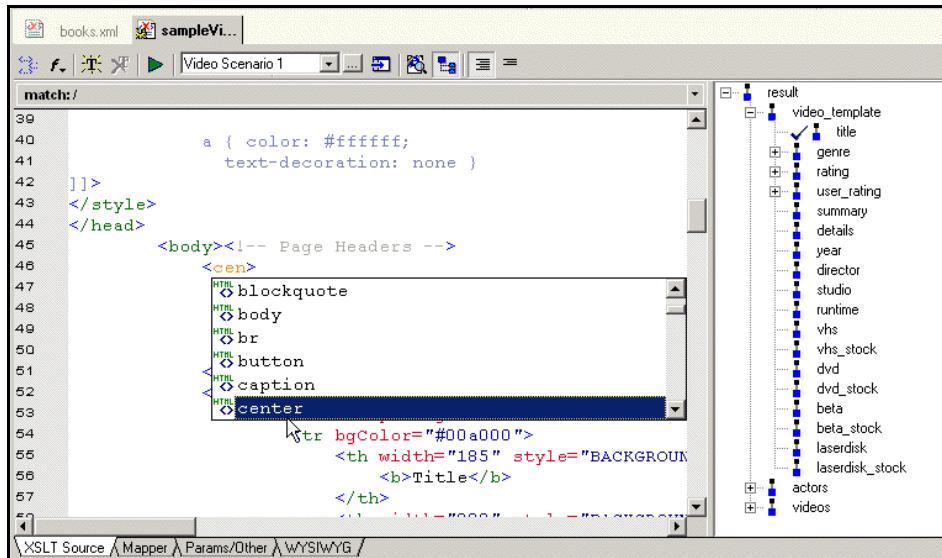
**Figure 203. Stylus Studio Displays XSLT Instructions for Quick Editing**

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 204. Sense:X Speeds Coding, Reduces Errors**

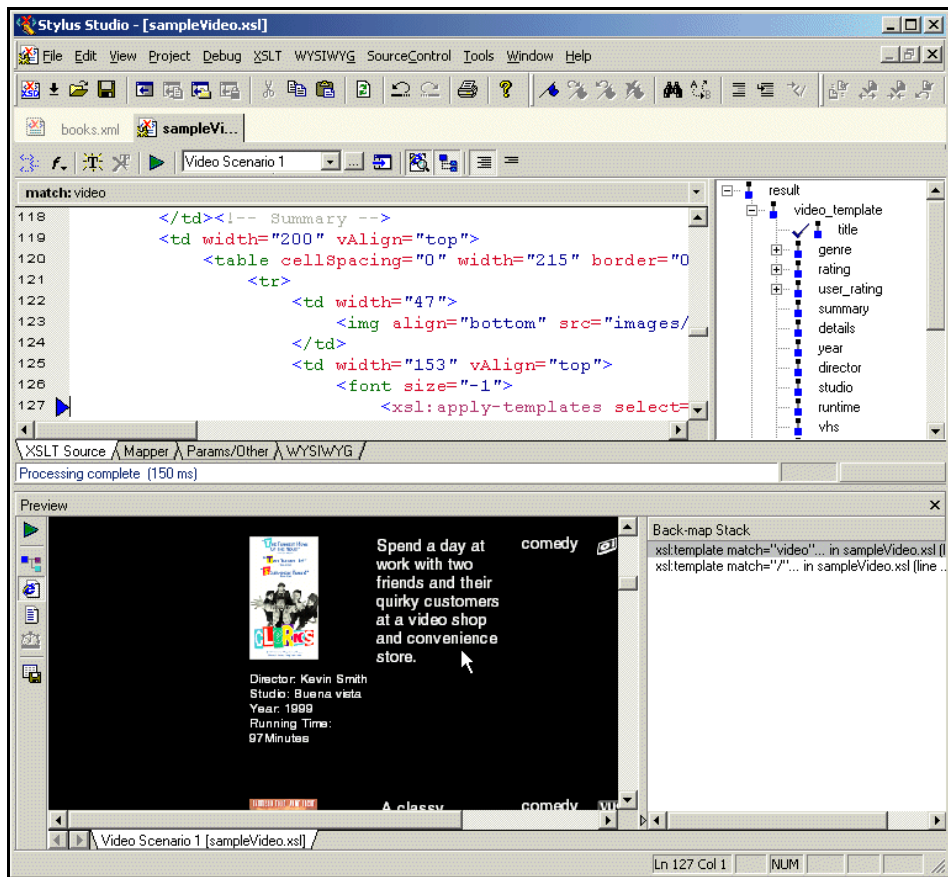
Sense:X also adapts to your document by suggesting more frequently used tags first. Valid XSLT and HTML tag names are color coded to improve readability.

## 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 205. 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 Edition.



## 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 the Xalan-J processor, or 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.

### Default Templates

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.

### About This Tutorial

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 340
- [Understanding How the Default Templates Work](#) on page 344
- [Editing the Template That Matches the Root Node](#) on page 348

- [Creating a Template That Matches the book Element](#) on page 349
- [Creating a Template That Matches the author Element](#) on page 350

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 26.

This tutorial duplicates some of the information in subsequent sections. For complete information, see the following topics:

- [Working with Stylesheets](#) on page 351
- [Working with Templates](#) on page 381

## 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: Text Editor**.

Stylus Studio displays a new untitled stylesheet and the **Scenario Properties** dialog box, and selects the text in the **Scenario Name** field.

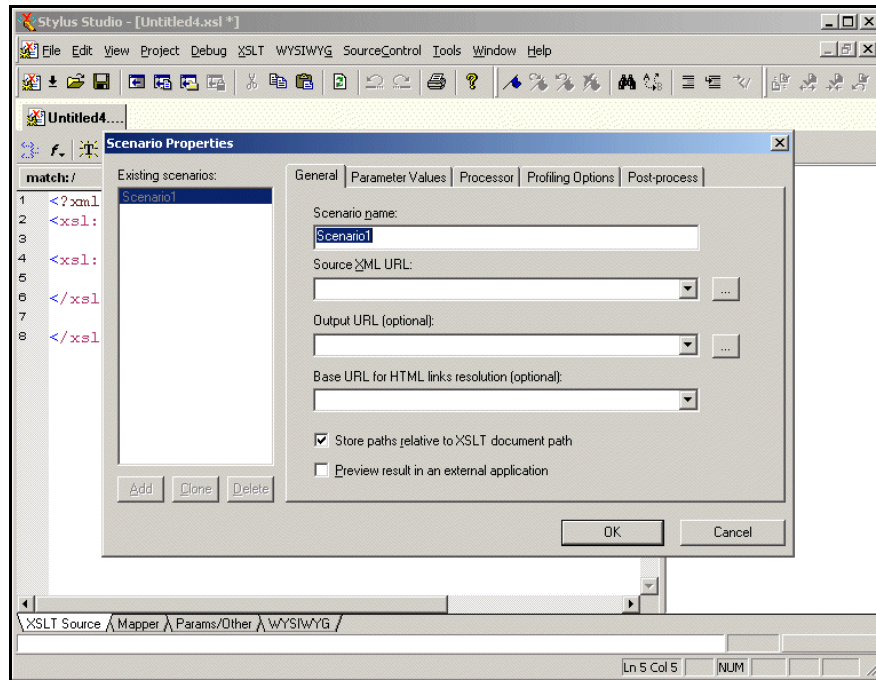

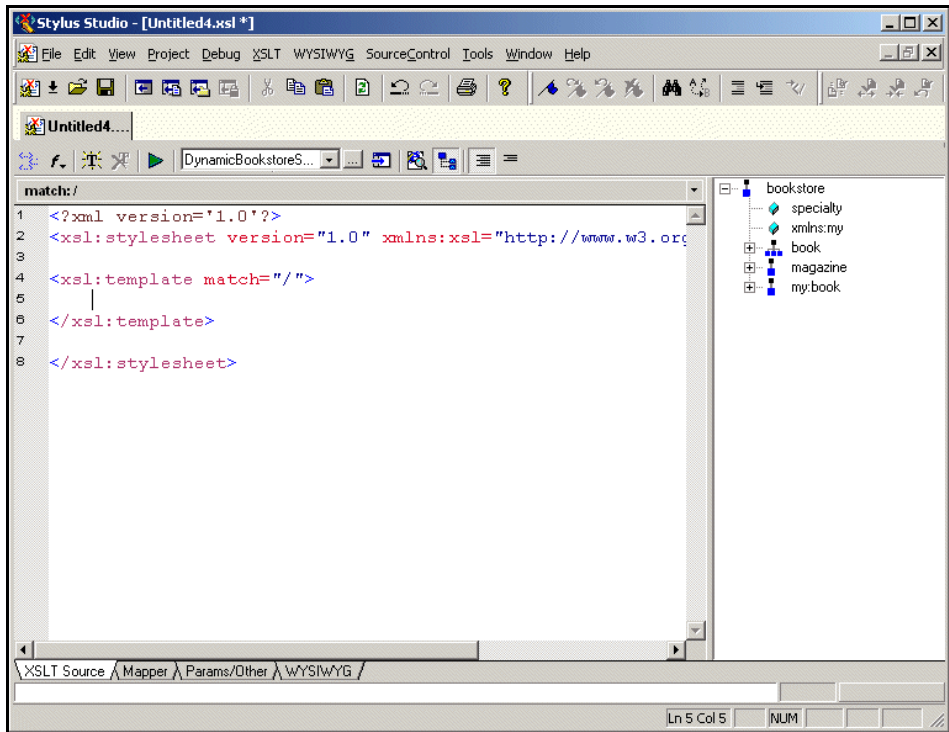


Figure 206. Scenarios Let You Easily Test Different XSLT/XML Pairs

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 207. 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 **bookstore.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 double-click **xsl:apply-templates**.
12. Type `/>`.
13. In the XSLT editor tool bar, click **Preview Result** . 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.

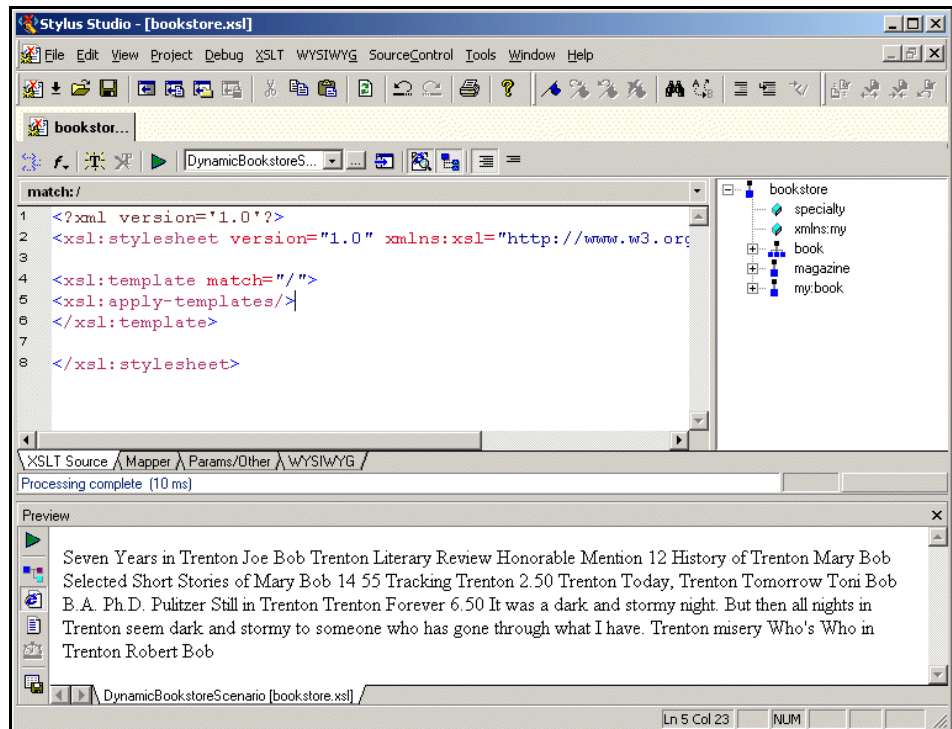



Figure 208. Default Templates Contain No Formatting Instructions

14. In the Stylus Studio tool bar, click **Save** .

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:

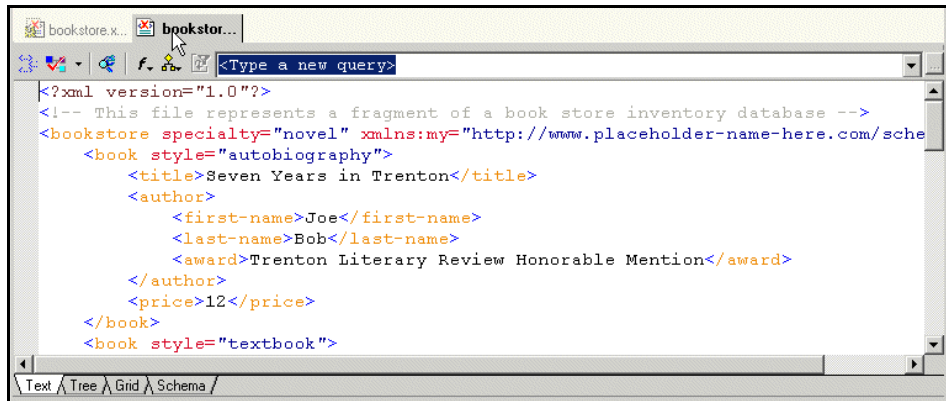
- [Instantiating the Template That Matches the Root Node](#) on page 344
- [Instantiating the Root/Element Default Template](#) on page 345
- [Instantiating the Text/Attribute Default Template](#) on page 346
- [Illustration of Template Instantiations](#) on page 347

### 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 `xsl:apply-templates` instruction. In this template, the `xsl: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 209. 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 `xs1:apply-templates` instruction. Like the template that matches the root node, the `xs1: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 `xs1: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:

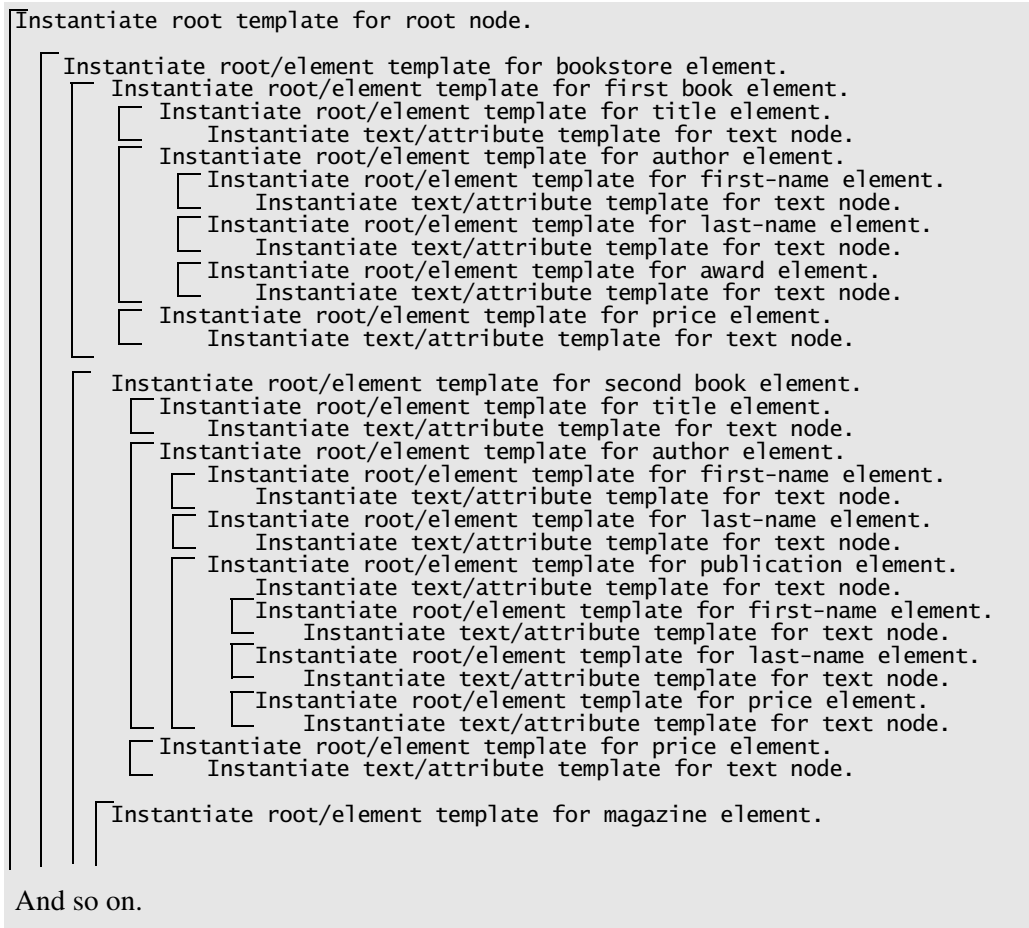
```
Seven Years in Trenton Joe
```

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:

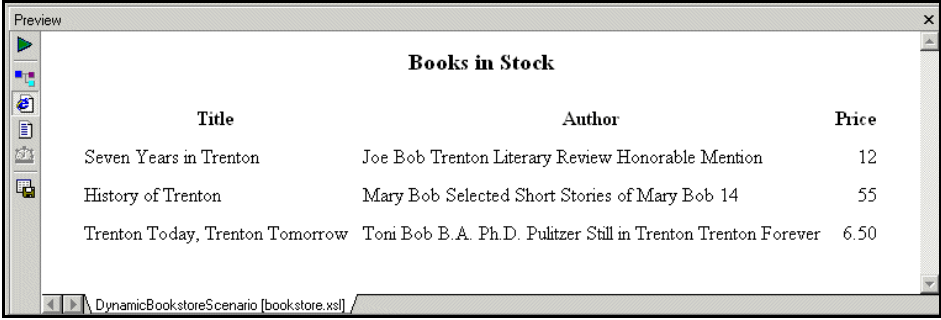
1. In the XSLT editor source document tree pane, expand the **bookstore** element.
2. Double-click the **book** element.

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 210](#):



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 210. Result of Applying XSLT**

In the book template, the `xs1: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:**

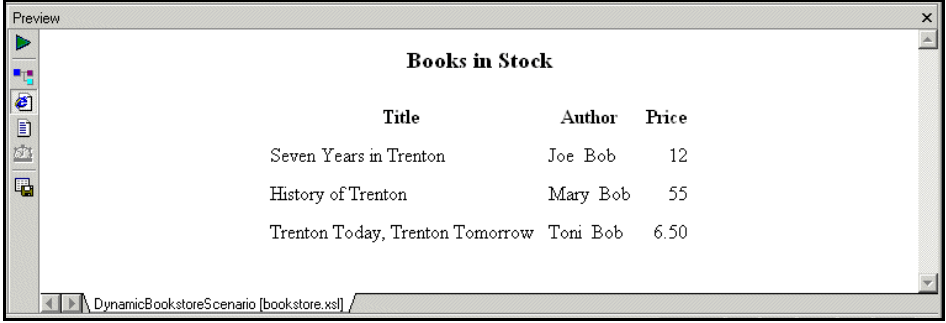
1. In the XSLT editor source document tree pane, expand the **book** element.
2. Double-click the **author** element.

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.

```
<xs1:value-of select="first-name"/>
 
<xs1: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 211](#).



Books in Stock		
Title	Author	Price
Seven Years in Trenton	Joe Bob	12
History of Trenton	Mary Bob	55
Trenton Today, Trenton Tomorrow	Toni Bob	6.50

**Figure 211. 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

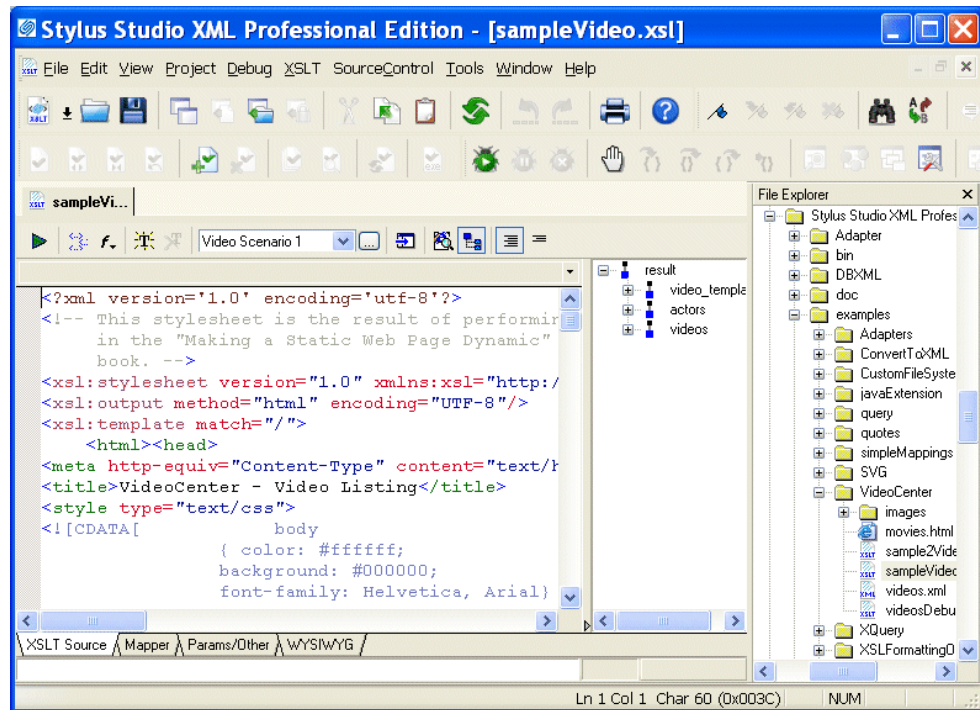
This section provides instructions for performing the various tasks involving stylesheets. See also “[Working with Templates](#)” on page 381. This section covers the following topics:

- [About the XSLT Editor](#) on page 352
- [Creating Stylesheets](#) on page 353
- [Creating a Stylesheet from HTML](#) on page 354
- [Specifying Stylesheet Parameters and Options](#) on page 354
- [Applying Stylesheets](#) on page 357
- [Applying a Stylesheet to Multiple Documents](#) on page 363
- [About Stylesheet Contents](#) on page 364
- [Updating Stylesheets](#) on page 365
- [Saving Stylesheets](#) on page 367

Also, Stylus Studio provides a number of tools that help you debug stylesheets. See “[Debugging Stylesheets](#)” on page 479.

## About the XSLT Editor

The XSLT editor, which displays a stylesheet when you open it, has four tabs at the bottom.



**Figure 212. XSLT Editor**

The **XSLT Source**, **Mapper**, and **Params/Other** tabs are always available. The **WYSIWYG** tab is available only when the stylesheet generates HTML.



The **WYSIWYG** tab is available only in Stylus Studio XML Enterprise Edition and Stylus Studio XML Professional Edition.

### 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: Text Editor**.

Stylus Studio displays the **Scenario Properties** dialog box.

*Alternative:* Select **File > New > XSLT: WYSIWYG**. This automatically sets the output method to HTML, and displays the **WYSIWYG HTML** editor. See “[Creating Stylesheets That Generate HTML](#)” on page 369.



The WYSIWYG HTML editor is available only in Stylus Studio XML Enterprise Edition and Stylus Studio XML Professional Edition.

*Alternative:* Select **File > New > XSLT: Mapper**. See “[Creating XSLT Using the XSLT Mapper](#)” on page 439.

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**.

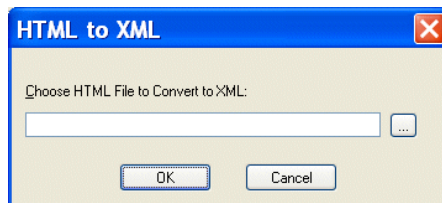
## Creating a Stylesheet from HTML

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 [Converting HTML to XML Documents](#) on page 163.

◆ **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.  
xxx pic needs to change



**Figure 213. 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

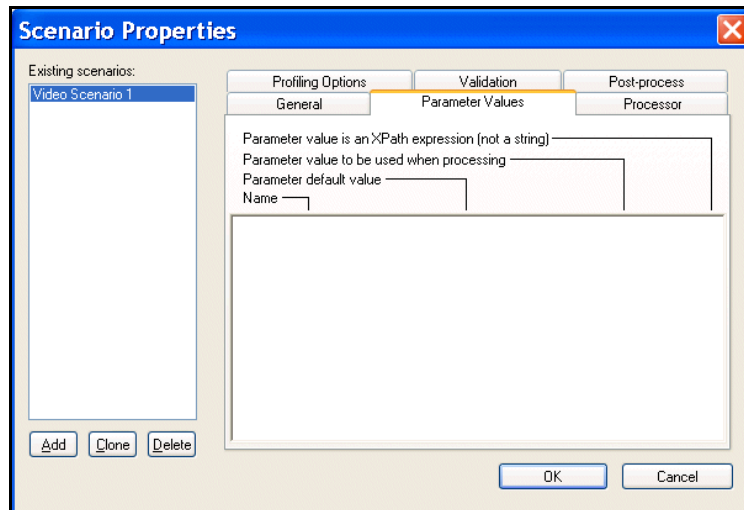
You can specify values for stylesheet parameters in the **Parameter Values** tab of the **Scenario Properties** dialog box.

◆ **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.

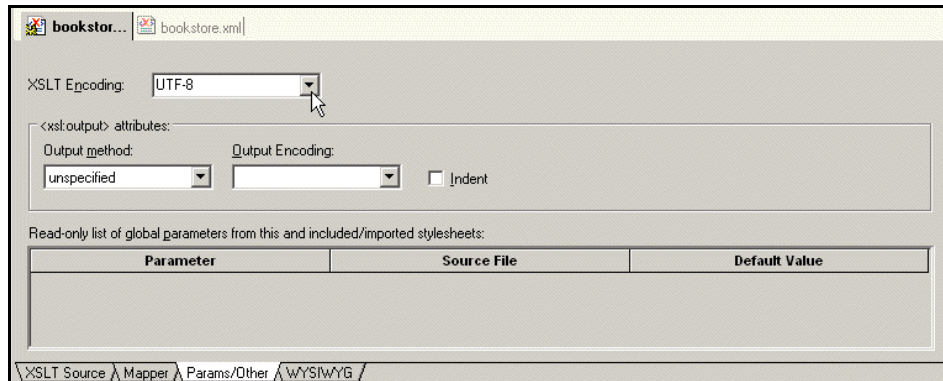


**Figure 214. XSLT Scenario Parameter Values Tab**

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 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 215. 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 360
- [“Cloning Scenarios”](#) on page 362
- [“Saving Scenario Meta-Information”](#) on page 362

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 357
- [Results of Applying a Stylesheet](#) on page 358
- [Applying Stylesheets to Large Data Sets](#) on page 360

**Tip** Stylus Studio provides a number of tools that help you debug stylesheets. See [“Debugging Stylesheets”](#) on page 479.



## 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.

### Multiple scenarios

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 148.

## 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.

Backmap  
stack  
window

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.

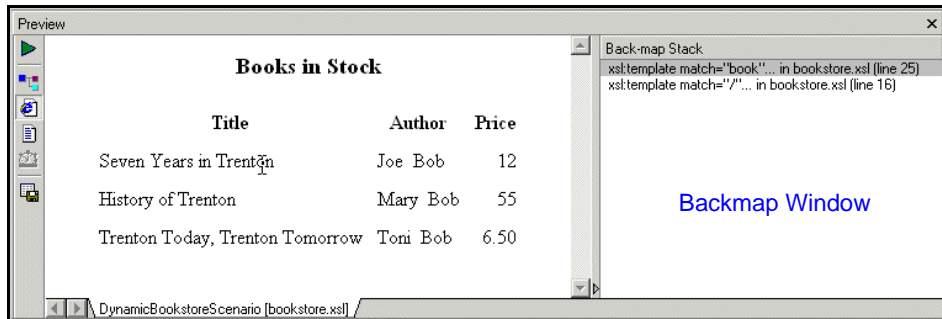


Figure 216. XSLT Backmap Window

Stylus Studio also flags the line in the stylesheet that contains the first instruction in the **Backmap Stack** window.

XML Tree  
view

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.

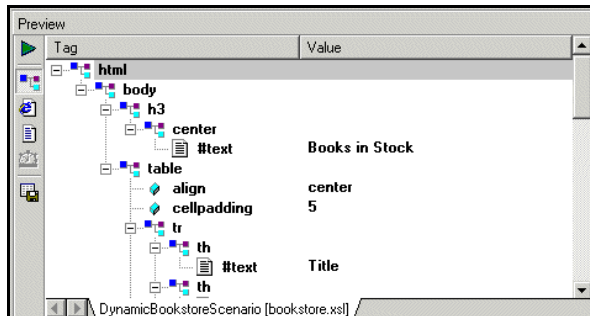



Figure 217. XML Tree View

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

Savig  
results

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 an External XSLT Processor”](#) on page 386.
10. If you want to specify any post-processing, click the **Post-process** tab. See [“Post-processing Result Documents”](#) on page 391.
11. To define another scenario, click **Add** and enter the information for that scenario. You can also copy scenarios. See [“Cloning Scenarios”](#) on page 362.
12. Click **OK**.


For more information, see [“Scenario Properties General Tab \(XSLT\)”](#) on page 980.

### 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 360.
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 954.
- 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 360.

#### ◆ **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 683.

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 364
- [Contents You Can Add](#) on page 365

### 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 383.



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 405. 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 179.

## 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.

Editing tools    The following sections describe the Stylus Studio editing tools:

- [Dragging and Dropping from Schema Tree into XSLT Editor](#) on page 365
- [Using Sense:X Automatic Tag Completion](#) on page 366
- [Using Sense:X to Ensure Well-Formed XML](#) on page 366
- [Using Standard Editing Tools](#) on page 367

## 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.

Double-clicking

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-xsl-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 926.

### 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 896.

## 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 912.

### 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.

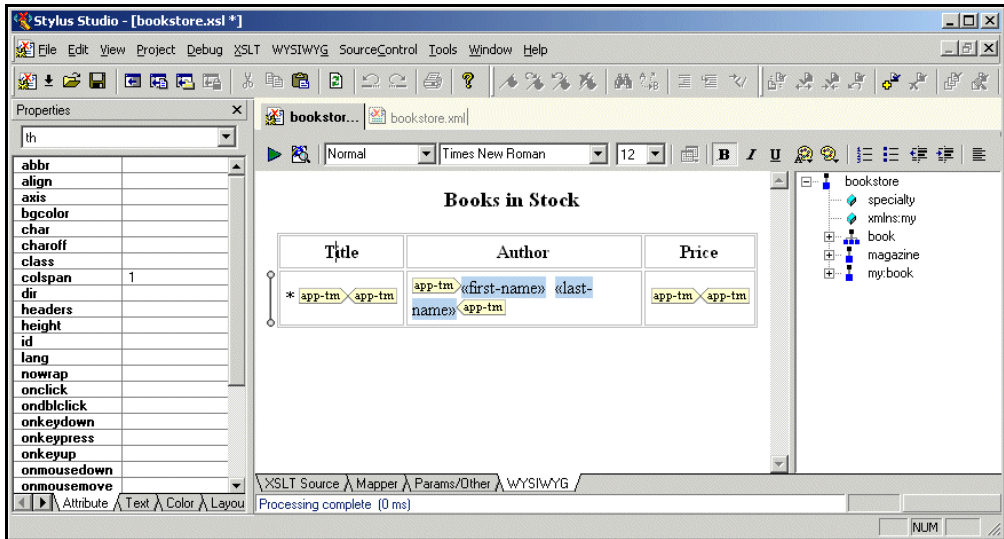
However, there is one situation in which Stylus Studio does not automatically use updated stylesheets. Suppose that multiple stylesheets are open in Stylus Studio. Each stylesheet generates a Web page, and the Web pages have links to each other. The stylesheets do not include or import each other. You 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.

## Creating Stylesheets That Generate HTML



The WYSIWYG HTML editor is available only in Stylus Studio XML Enterprise Edition and Stylus Studio XML Professional Edition.

The WYSIWYG HTML editor allows you to design stylesheets that generate HTML in a way that is easier than typing all required XSLT instructions.



**Figure 218. WYSIWYG HTML Editor for XSLT**

Stylus Studio synchronizes the **XSLT Source** and **WYSIWYG** tabs in the XSLT editor. Any updates you make in one tab are automatically reflected in the other tab. This allows you to go back and forth between these two views as needed. You might find that it is easier to insert some XSLT instructions directly, and then return to the **WYSIWYG** tab.

In the **WYSIWYG** tab, Stylus Studio displays a tree that represents the XML source document that the stylesheet operates on. It is important to note that when the XML source document is associated with a schema, the tree still represents the XML source document. It does not represent the associated schema.

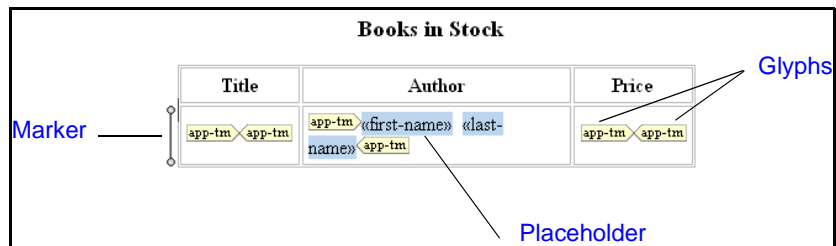
The stylesheet that you design in the **WYSIWYG** tab can generate HTML or XHTML. Consequently, the output method must be HTML or XML. The output method cannot be text or unspecified.

This section covers the following topics:

- [Descriptions of WYSIWYG Terms](#) on page 370
- [Inserting Contents in the HTML Editor](#) on page 371
- [Displaying a Repeating Element in the HTML Editor](#) on page 372
- [Adding Conditional Processing in the HTML Editor](#) on page 372
- [Instantiating Templates in the HTML Editor](#) on page 375
- [Specifying Properties and Attributes in the HTML Editor](#) on page 376

## Descriptions of WYSIWYG Terms

The following terms are used to describe parts of the **WYSIWYG** tab:



**Figure 219. Parts of HTML Diagram in the WYSIWYG Tab**

- *placeholder* - an element or attribute name in double angle brackets. It represents dynamic text.
- *marker* - a small image to the left of a placeholder or table row. Hold the cursor over a marker to see the XSLT instruction it represents and the context in which it operates.
- *glyphs* - pairs of images that specify choose, if, or repeat. The corresponding XSLT instruction operates on the text represented by the enclosed placeholder or element.

To remove the glyphs, select the enclosed item, and right-click it to display a pop-up menu. Click **Unwrap selected element**.



## Inserting Contents in the HTML Editor

The **WYSIWYG** tab provides an HTML editor that operates as you would expect. You can type text and select it to format it. The usual formatting tools appear in the **WYSIWYG** tool bar. Select the text you want to operate on and then click the tool that provides the feature you want.

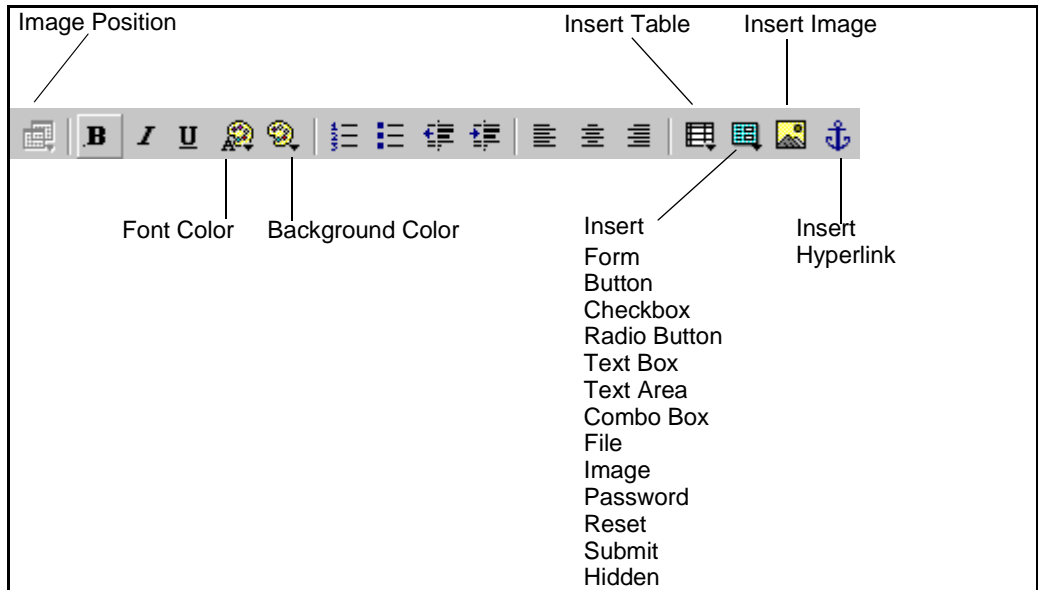




Figure 220. WYSIWYG Tool Bar


### Image Position


**Image Position**  allows you to specify whether the position of an image is fixed or dependent on the contents that come before it. Click the image for which you want to set this position attribute. Then click **Make Absolute** if you want the image to always appear in the same location regardless of where the image is in the HTML source. You can also specify whether the image is movable. When **Movable** is selected, you can move the image to a new location. The image always appears in the new location unless you explicitly move it again.

## Displaying a Repeating Element in the HTML Editor

In the source tree diagram, repeating elements are identified by .

There are several ways to define a repeating element in the HTML canvas:

- Drag a repeating element from the XML source tree and drop it onto the HTML canvas. In the pop-up menu that appears, click **xsl:for-each**.
- Click **Insert Table** , drag and drop a repeating element onto a table cell, and click **Make Repeating** in the pop-up menu.

When you define a repeating element, Stylus Studio displays a marker () to the left of the element. When you click this marker, Stylus Studio displays the properties of the elements that are displayed by the `xsl:for-each` instruction.

When you select **Add Table** from the pop-up menu, Stylus Studio always creates a tables with two columns. You can, of course, delete one of the columns if you do not need it.

When you drag a repeating element onto a placeholder that is already a repeating element, Stylus Studio displays a pop-up menu that includes the **Reset repeating XPath** option. Select this option if you want to reset the context for the repeating element. The new context is the element that you dragged in.


## Adding Conditional Processing in the HTML Editor

Conditional processing can be on a single item, a repeating item or a row.

- To specify conditional processing on a single or repeating item, right-click the placeholder and select **Conditional Processing** from the pop-up menu.
- To specify conditional processing for an entire row, right-click in the row, and select **Conditional Processing (Table Row)** from the pop-up menu.

You can then specify whether you want to insert an `xsl:choose` or `xsl:if` instruction.

Only one branch of a conditional instruction can be active at any one time. Consequently, the HTML canvas might haze or hide glyphs and placeholders. If you want to operate on a placeholder that is not visible, you must change the active branch.

When you specify conditional processing on a table row, a marker () appears to the left of the table row. You can move the cursor over the marker to display a message that indicates the active branch. Click the marker to display the properties for the active conditional branch associated with the marker you clicked.

This section covers the following topics:

- [Specifying Choose Conditional Processing in the HTML Editor](#) on page 373
- [Specifying If Conditional Processing in the HTML Editor](#) on page 374

## Specifying Choose Conditional Processing in the HTML Editor

At any point in time only one branch of an `xs1:choose` instruction can appear in the HTML canvas. This is the active branch. When you specify formatting or properties for a branch of an `xs1:choose` instruction, it applies to only the elements in that branch. When you move the cursor over a **choose** glyph, Stylus Studio displays a message that specifies which branch is the active branch.

Use the **Choose Manager** to define the test conditions in the `xs1:choose` instruction. Use the **Properties** window to change the active branch.

### Choose Manager

◆ **To specify `xs1:choose` processing:**

1. Right click on the placeholder for which you want to define the `xs1:choose` instruction.
2. In the shortcut menu that appears, select **Conditional Processing > Add Choose**. Stylus Studio displays the **Choose Manager** dialog box.

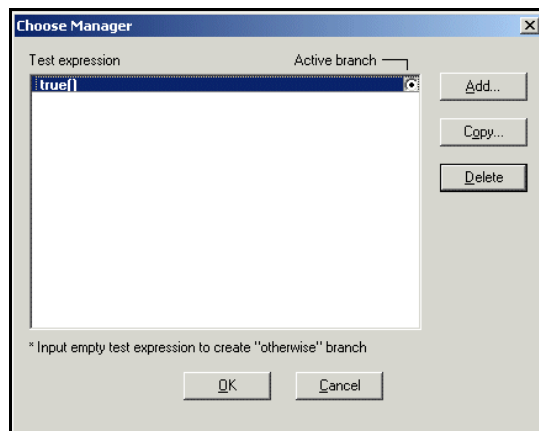


Figure 221. Choose Manager Dialog Box

3. Click **Copy** to define a test condition, which must be an XPath expression. You can define as many conditions as you need. Each condition you add corresponds to an `xs1:when` instruction.
4. If you do not need it, select the **true()** condition and delete it.
5. To specify an `xs1:otherwise` instruction, add a condition and leave it blank.
6. Click **OK**.  
Stylus Studio creates an `xs1:choose` instruction that wraps the placeholder or row that you selected.

### Change active branch

#### ◆ To change the active branch:

1. In the HTML canvas, click a **choose** glyph for the active branch you want to change. Stylus Studio displays the properties for the `xs1:choose` instruction you clicked.
2. In the **Properties** window, click the **when** field.
3. Click the down arrow to display the drop-down menu.
4. Click the branch you want to be active.

For a new `xs1:choose` instruction, the **Choose Manager** displays one test expression: `true()`. This is the initial, required, `xs1:when` instruction.

**Tip** Stylus Studio displays only what you can operate on based on the active conditional branch. In other words, if processing at a particular point cannot operate on certain elements, then those elements do not appear in the HTML canvas. If you make an `xs1:choose` branch active, and the XSLT processor cannot reach it, Stylus Studio displays a **false()** glyph.

### Specifying If Conditional Processing in the HTML Editor

#### ◆ To specify `xs1:if` conditional processing:

1. Select the placeholder for which you want to define the `xs1:if` instruction.
2. In the pop-up menu that appears, select **Conditional Processing > Add If**. Stylus Studio displays the **Input** dialog box.

3. In the **Test Condition** field, specify an XPath expression. Stylus Studio evaluates it in the context of the placeholder to which you are adding the `xsl:if` instruction.
4. Click **OK**.

You can specify multiple `xsl:if` instructions on the same placeholder.

## Instantiating Templates in the HTML Editor

### ◆ To define a template in the WYSIWYG tab:

1. Drag an element or attribute from the source tree and drop it on the HTML canvas.
2. In the pop-up menu that appears, click **xsl:apply-templates**.  
Stylus Studio creates a template that matches the node you dragged in. You can select the placeholder, and apply formatting. Stylus Studio adds this information to the template.

## Calling a Named Template

If you have one or more named templates in your stylesheet, or if you want to define a named template, you can add an `xsl:call-template` instruction.

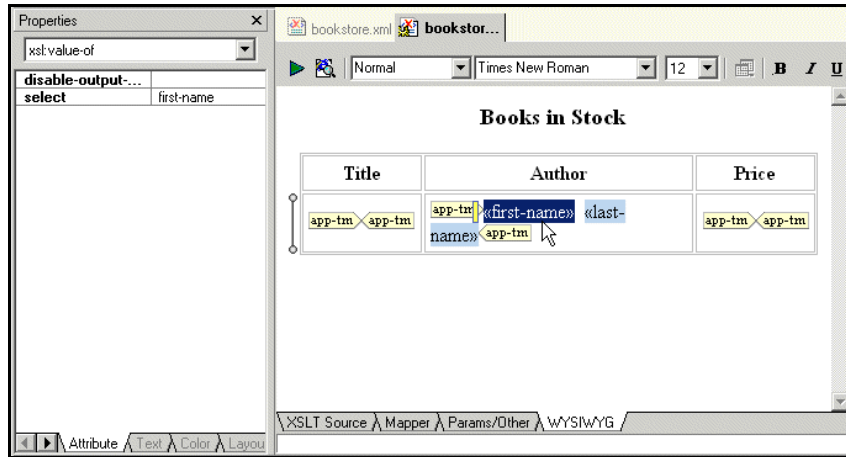
### ◆ To call or create a named template in the WYSIWYG tab:

1. Do one of the following:
  - Right-click in the HTML canvas and select **Add call-template** from the pop-up menu. You can then drag contents to between the `xsl:call-template` glyphs.
  - Select a placeholder or element in the HTML canvas. Right-click to display the pop-up menu, and select **Add call-template**.
2. Click the name of the template you want to instantiate, or click **Create new template** to create a new named template.

**Note** If you delete `xsl:apply-templates` glyphs, the corresponding template is still defined in the XSLT source. You must delete this definition in the source when you want to define new contents for the corresponding template in the HTML editor.

## Specifying Properties and Attributes in the HTML Editor

The **Properties** window displays the properties for the location you click in the HTML canvas. The immediately visible properties are for the element that directly contains the location you clicked.



**Figure 222. Properties Window Displays Selected HTML**

In the **Properties** window, click the down arrow to display a list of the elements that enclose the element that directly contains the location you clicked. The elements are listed from innermost to outermost. Click an element to display its properties.

## 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. The syntax supported is a subset of that supported by LotusXSL/Apache Xalan.

This section covers the following topics:

- [Using an Extension Function in Stylus Studio](#) on page 377
- [Basic Data Types](#) on page 378
- [Declaring an XSLT Extension Function](#) on page 378
- [Working with XPath Data Types](#) on page 379
- [Declaring an Extension Function Namespace](#) on page 379

- [Invoking Extension Functions](#) on page 380
- [Finding Classes and Finding Java](#) on page 380
- [Debugging Stylesheets That Contain Extension Functions](#) on page 380

## 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 `xsl: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 33](#):

**Table 33. 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.*;
public class NumberUtils
{
    public Object Average(NodeList n1)
    {
        double nSum = 0;
        for (int i = n1.getLength() - 1; i >= 0; i--)
        {
            nSum +=
                Double.valueOf(n1.item(i).
                    getNodeValue()).doubleValue();
        }
        return new Double(nSum / n1.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 Edition and Stylus Studio XML Professional Edition.

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

- Xalan-J XSLT processor
- Saxon processor

The Xalan-J and Saxon processors do 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 344.

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 381
- [Using Stylus Studio Default Templates](#) on page 383
- [Creating Templates](#) on page 385
- [Applying Templates](#) on page 386
- [Updating Templates](#) on page 386
- [Deleting Templates](#) on page 386

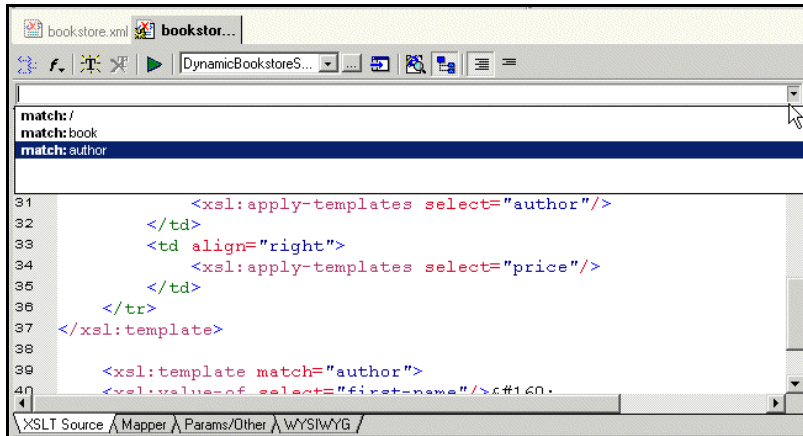
### Viewing Templates

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 223. 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 383
- [About the Root/Element Built-In Template](#) on page 384
- [About the Text/Attribute Built-In Template](#) on page 384

## 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 `xmlns:apply-templates` instruction. The `xmlns: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 `xmlns: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:  

```
<b><xsl:value-of select="."/></b>
```
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 `<b>` and `</b>`. 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 `xs1: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 323.

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 365.

### 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 an External XSLT Processor

In addition to a built-in XSLT processor, Stylus Studio includes several third-party XSLT processors, including Xalan-J, MSXML .NET, and Saxon (6 and 8). Note, however, that only the following XSLT processors support Stylus Studio stylesheet debugging and back-mapping functionality:

- Stylus Studio’s built-in XSLT processor
- XalanJ
- Microsoft .NET
- Saxon 6 and 8



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 discusses the following topics:

- [How to Use an External Processor](#) on page 387
- [Setting Default Options for Processors](#) on page 388

## How to Use an External 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 an external XSLT processor, output from the processor appears in the **Preview** window.

### ◆ To use an external 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 an external 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 360.
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 processor 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**.

### Passing Parameters

To pass parameters to the Stylus Studio built-in XSLT processor or to the Xalan-J, 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

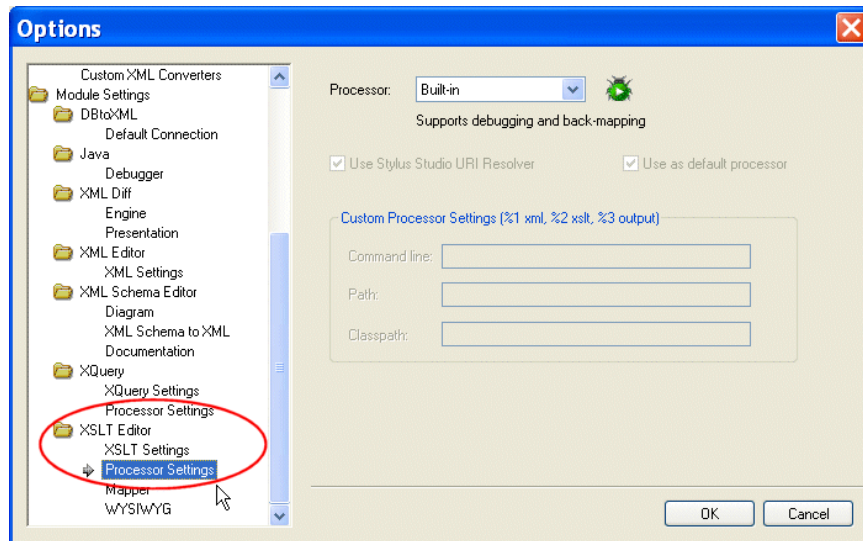
If you want, you can set default values for XSLT processor options and designate a processor other than the built-in processor as the default processor used whenever you create an XSLT scenario.

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 224. Options for XSLT Processors**

3. Select the processor for which you want to specify default settings from the **Processor** drop-down list.
4. If you want this processor to be used as the default processor for all XSLT scenarios, click the **Use as default processor** check box.
5. Click **OK**.


## Validating Result Documents

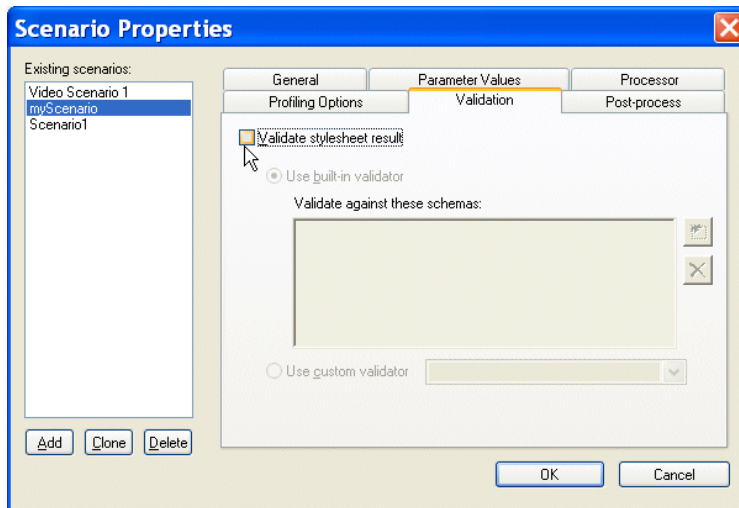
You can optionally validate the XML document that results from XSLT processing. You can validate using the

- 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, Xerces-J, 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 225. 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 392.

### ◆ 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 393
- [Troubleshooting FOP Errors](#) on page 393
- [Viewing the FO Sample Application](#) on page 394
- [Deploying Stylesheets That Generate FO](#) on page 396
- [Using Apache FOP to Generate NonPDF Output](#) on page 397

**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

**Tip** Initially, develop the stylesheet with Stylus Studio’s internal XSLT processor. It gives better error messages than the Xalan-J processor. When the transformation and display appear to work with the built-in XSLT processor, try using the Xalan-J processor in Stylus Studio. The Xalan-J processor is the one that the FOP uses when you apply this stylesheet from a command line.

◆ **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 360.
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 391.
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.

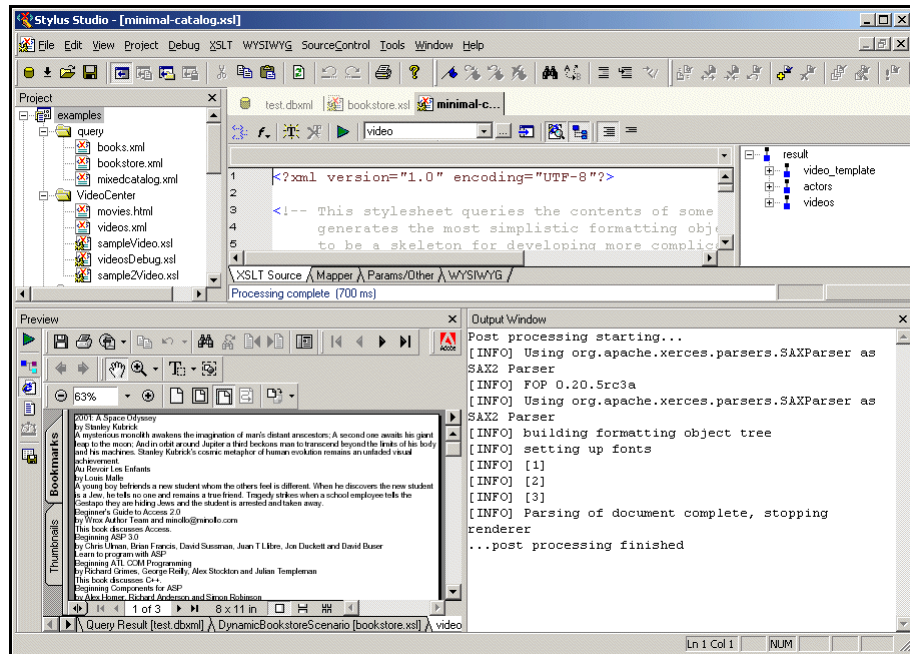
The video scenario has already been defined. In the Post-process tab of the Scenario Properties dialog box, **Postprocess with Apache FOP** is selected.

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



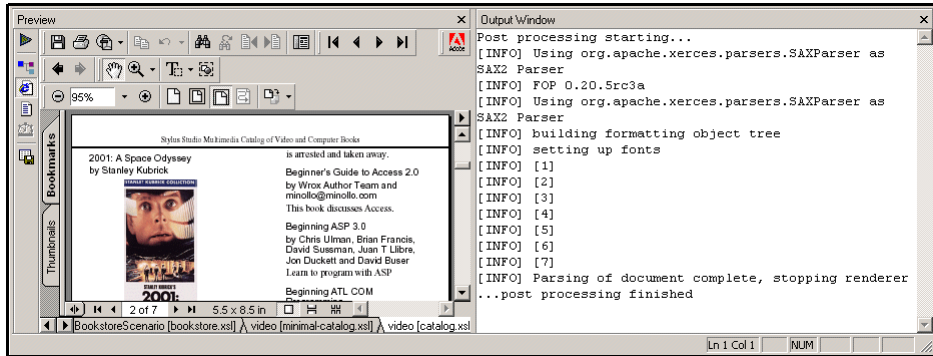
2. Click **Preview Result** . As you can see, the **Output Window** shows some post-processing information messages.



**Figure 226. 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.xml` stylesheet.

5. Click **Preview Result**  .

**Figure 227. 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;C:\Program
Files\StylusStudio\bin\xalan.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 360. 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 34. FOP Output Options**

<b>Setting</b>	<b>Output</b>
-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.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 `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

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.

This section covers the following topics:

- [“Scenario Settings”](#) on page 399
- [“Java Code Generation Settings”](#) on page 401
- [“How to Generate Java Code for XSLT”](#) on page 402
- [“Compiling Generated Code”](#) on page 403

**Tip** You can also generate Java code for XQuery. See [“Generating Java Code for XQuery”](#) on page 739.

### 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 35. Scenario Settings that Affect Java Code Generation**

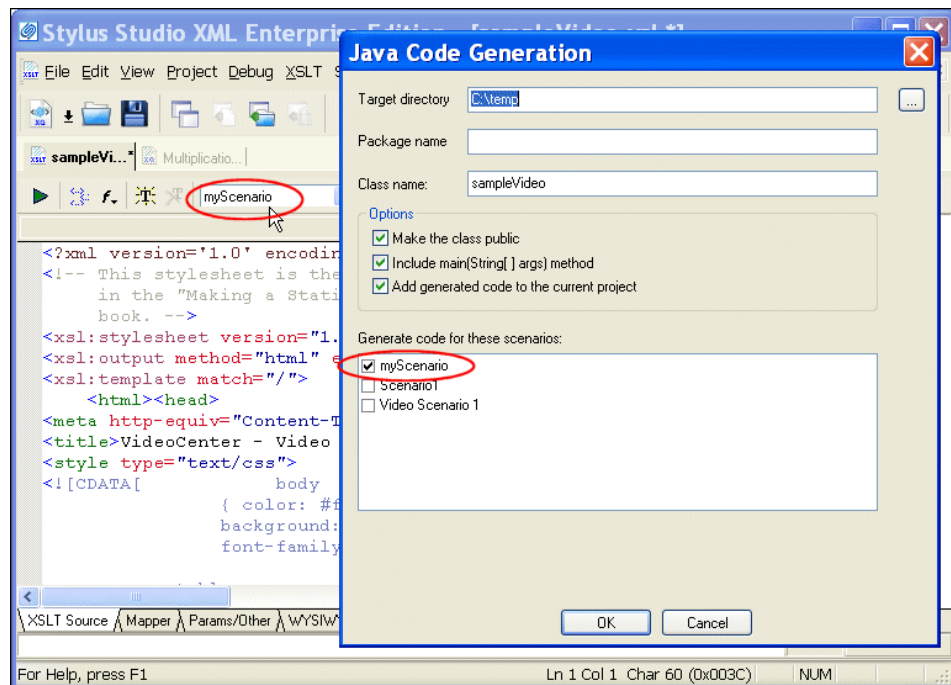
<i>Tab</i>	<i>Comment</i>
General	The Java Code Generation wizard uses only the <b>Source XML URL</b> , and the <b>Output URL</b> field, if specified. All other properties on this page are ignored.
Parameter Values	Default parameter values and parameters in the form of XPath expressions are ignored.
Processor	You must use a Saxon or XalanJ processor. The Stylus Studio URI Resolver is also used to resolve non-standard URIs if you select the check box on the <b>Processor</b> page.
Profiling Options	Ignored.

**Table 35. Scenario Settings that Affect Java Code Generation**

<i>Tab</i>	<i>Comment</i>
Validation	Validation is optional. If you choose to validate XSLT output, the Java Code Generation wizard always uses the Xerces-J processor, regardless of the validator you specify on the <b>Validation</b> tab. If you want to specify external schemas for validation purposes, click <b>Use built-in validator</b> . Note that the Xerces-J 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 228](#).

**Figure 228. 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

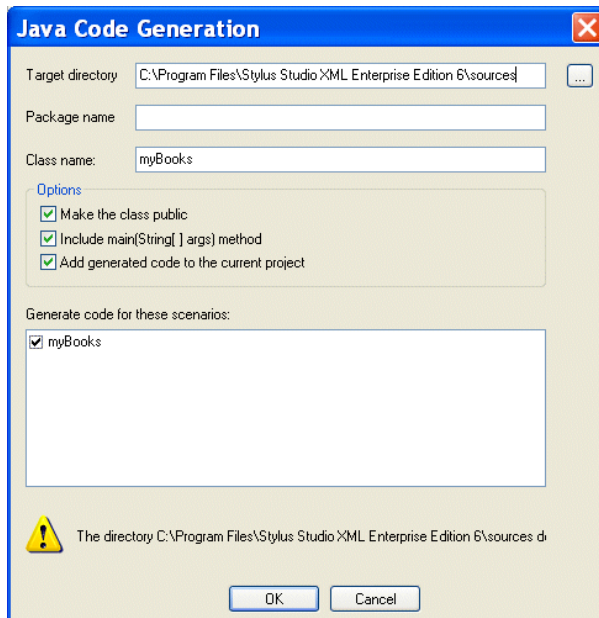
All of these options are selected by default.

**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 399 for more information.
2. Select **XQuery > Generate Java Code** from the Stylus Studio menu. The **Generate Java Code** dialog box appears.



**Figure 229. 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 401 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 sXSLT, 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

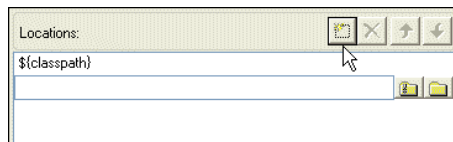
### ◆ To modify the Stylus Studio classpath:

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 230. Entry Field for JAR File Classpath**

5. Click the browse jar files button ().  
Stylus Studio displays the **Browse for JAR Files** dialog box.




**Figure 231. 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.

---

## XSLT Instructions Quick Reference

This section provides a quick reference for the XSLT instructions supported by the Stylus Studio XSLT processor. For additional information, see the W3C XSLT Recommendation at <http://www.w3.org/TR/xslt>. The instructions described in this section are listed below.

- [xsl:apply-imports](#) on page 406
- [xsl:apply-templates](#) on page 406
- [xsl:attribute](#) on page 407
- [xsl:attribute-set](#) on page 409
- [xsl:call-template](#) on page 411
- [xsl:choose](#) on page 411
- [xsl:comment](#) on page 412
- [xsl:copy](#) on page 413
- [xsl:copy-of](#) on page 414
- [xsl:decimal-format](#) on page 414
- [xsl:element](#) on page 416
- [xsl:fallback](#) on page 417
- [xsl:for-each](#) on page 417
- [xsl:if](#) on page 419
- [xsl:import](#) on page 420
- [xsl:include](#) on page 420
- [xsl:key](#) on page 421
- [xsl:message](#) on page 422
- [xsl:namespace-alias](#) on page 423
- [xsl:number](#) on page 423
- [xsl:otherwise](#) on page 425
- [xsl:output](#) on page 425
- [xsl:param](#) on page 427
- [xsl:preserve-space](#) on page 428
- [xsl:processing-instruction](#) on page 428
- [xsl:sort](#) on page 429
- [xsl:strip-space](#) on page 431

- [xsl:stylesheet](#) on page 432
- [xsl:template](#) on page 432
- [xsl:text](#) on page 434
- [xsl:transform](#) on page 435
- [xsl:value-of](#) on page 435
- [xsl:variable](#) on page 436
- [xsl:when](#) on page 437
- [xsl:with-param](#) on page 437

### 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 429.

**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: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 `xs1:choose` element contains one or more `xs1:when` elements followed by zero or one `xs1:otherwise` element. Each `xs1:when` element contains a required `test` attribute, whose value is an expression. Each `xs1:when` and `xs1:otherwise` element contains a template.

When the XSLT processor processes an `xs1:choose` element, it starts by evaluating the expression in the first `xs1: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 `xs1:when` element. If the result is `false`, the XSLT processor evaluates the expression in the next `xs1:when` element.

The XSLT processor instantiates the template of only the first `xs1:when` element whose test expression evaluates to `true`. If no expressions evaluate to `true` and there is an `xs1:otherwise` element, the XSLT processor instantiates the template in the `xs1:otherwise` element.

If no expressions in `xs1:when` elements are `true` and there is no `xs1:otherwise` element, the `xs1:choose` element has no effect.

**Tip** You can create an `xs1:choose` element automatically using the XSLT mapper.

### `xs1:comment`

Adds a comment node to the result tree.

### Format

```
<xs1:comment>  
  comment_text  
</xs1: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 `xs1: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.

### 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 `xs1:decimal-format` correspond to the methods on the JDK `DecimalFormatSymbols` class. For each get/set method pair, there is an attribute defined for the `xs1: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 429.

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: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 411.

### 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: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 681.

**Restriction** 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 411.

## xsl:output

Specifies the output for the result tree.

### Format

```
<xsl:output attribute_list />
```

### Description

The `xsl: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 `xsl:output` instruction has no effect.

If you specify the `xsl:output` instruction, the XSLT processor outputs the result tree according to your specification. If you specify it, the `xsl: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 `xsl: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`.

A stylesheet can include multiple `xs1:output` elements. The XSLT processor effectively merges multiple `xs1:output` elements into one `xs1:output` element. If there are multiple values for the same attribute, the XSLT processor uses the last specified value.

Ignored  
attributes

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: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 437.

### 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:sort

Sorts the set of nodes selected by an `xsl:apply-templates` or `xsl:for-each` instruction.

### Format

```
<xsl:sort
  [select="expression"]
  [optional_attribute]/>
```

### Description

The `xsl:sort` instruction must be the child of an `xsl:apply-templates` or `xsl:for-each` instruction. Each `xsl:apply-templates` and `xsl:for-each` instruction can contain more than one `xsl:sort` instruction. The first `xsl:sort` child specifies the primary sort key. The second `xsl:sort` child, if any, specifies the secondary sort key, and so on.

When an `xsl:apply-templates` or `xsl:for-each` element contains an `xsl:sort` instruction, the selected nodes are processed in the order specified by the `xsl:sort` instructions. When `xsl:sort` elements are in an `xsl: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 `xsl:apply-templates` or `xsl: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 `xsl: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 `xsl:sort order` at run time by using an attribute value template. For example:

```
<xsl: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 `xs1: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 381.

**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 432.

## 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.

#### Scope

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 411.

## xsl:with-param

Passes a parameter value to a template.

### Format

```
<xsl:with-param
  name = "parameter_name"
  [select = "expression1"]
  [expr = "expression2"]>
  [parameter_value]
</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 `xsl:with-param` element must be a child of `xsl:apply-templates` or `xsl:call-template`.

You can specify the `xsl:with-param` instruction in `xsl:call-template` and `xsl:apply-template` instructions.

### Example

Suppose you specify the following parameter for a template:

```
<xsl:template name="Appendix">
  <xsl:param name = "heading"> 1. </xsl-param>
  ...
</xsl:template>
```

You can pass another value for this variable as follows:

```
<xsl:call-template name = "Appendix">
  <xsl:with-param name = "heading"> A. </xsl:with-param>
</xsl:call-template>
```

## Chapter 5    **Creating XSLT Using the XSLT Mapper**

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 64.

This chapter covers the following topics:

- [“Overview of the XSLT Mapper”](#) on page 439
- [“Source Documents”](#) on page 445
- [“Target Structures”](#) on page 451
- [“Mapping Source and Target Document Nodes”](#) on page 454
- [“Working with XSLT Instructions in XSLT Mapper”](#) on page 456
- [“Processing Source Nodes”](#) on page 464
- [“Creating and Working with Templates”](#) on page 470
- [“Creating an XSLT Scenario”](#) on page 472

### **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.

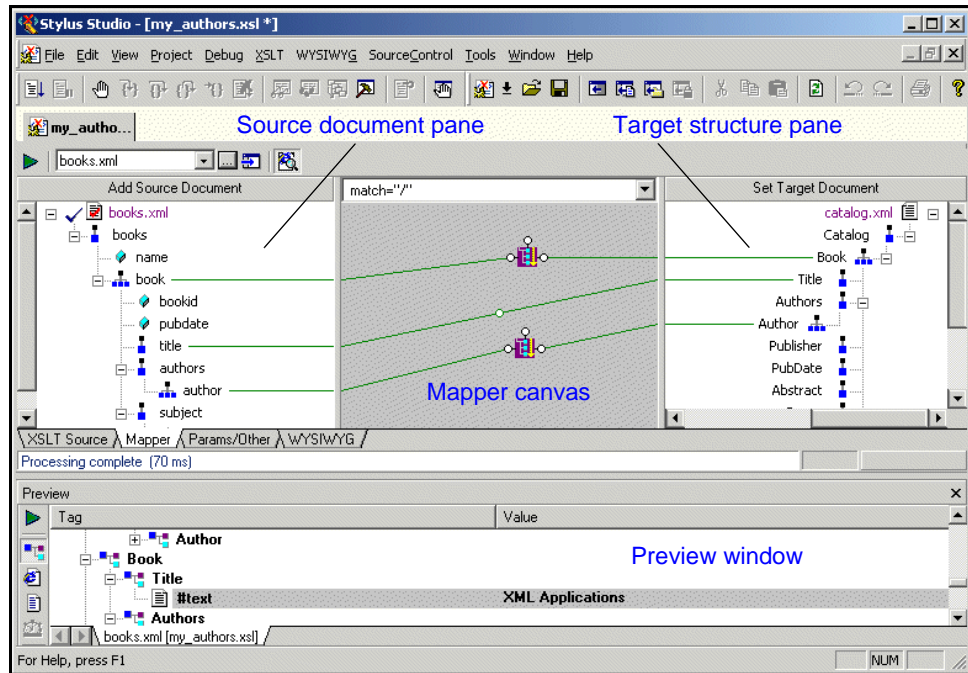



Figure 232. Example of XSLT Mapper

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 232). 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 441
- [Graphical Support for Common XSLT Instructions and Expressions](#) on page 442
- [Setting Options for the XSLT Mapper](#) on page 443
- [Ensuring That Stylesheets Output Valid XML](#) on page 444

### 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`.

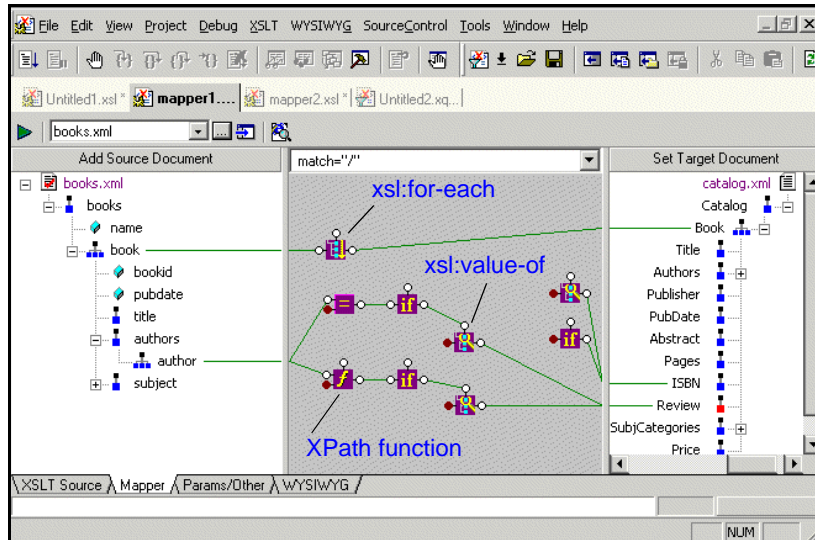
Now suppose you want to apply a stylesheet to `catalog.xml` and output an XML file that has a structure similar to `books.xml`. To do this, you must use the XSLT mapper to create a second stylesheet. This time, `catalog.xml` is the source document and `books.xml` is the destination document. The result 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 233](#):



**Figure 233. 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 `xsl: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 456 and [“Processing Source Nodes”](#) on page 464 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.

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.

### 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.

In this section


This section covers the following topics:

- [“Choosing Source Documents”](#) on page 445
- [“Source Documents and XML Instances”](#) on page 446
- [“How to Add a Source Document”](#) on page 448
- [“How to Remove a Source Document”](#) on page 450
- [“How Source Documents are Displayed”](#) on page 450

### 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.

For more information

See [“Source Documents and XML Instances”](#) on page 446 to learn more about how Stylus Studio treats source documents. See [“Creating an XSLT Scenario”](#) on page 472 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:

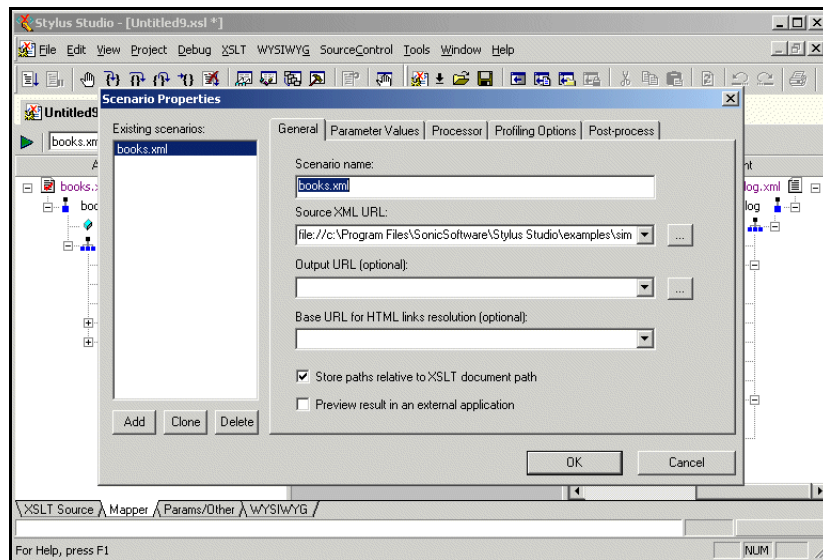
```
document("file:///c:/Program Files/Stylus Studio/examples/simpleMappings/catalog.xml")/books/book
```

#### 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 `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 234. 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 472.

- 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 36. 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 <b>Source XML URL</b> 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 <a href="#">“How to change a source document association”</a> on page 448.
	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

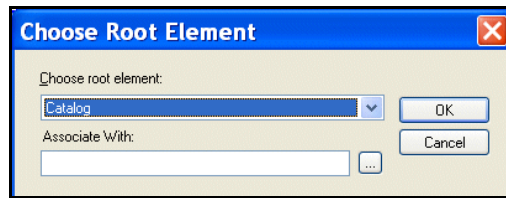
◆ **To add a source document to XSLT mapper:**

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 235. 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 456.

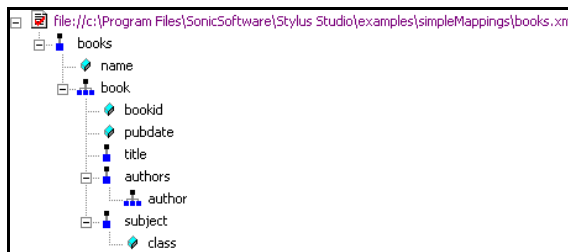
◆ **To remove a source document from the XSLT mapper:**

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 456 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 446 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 236. 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 37. Document Structure Symbols**

<i>Symbol</i>	<i>Meaning</i>
	Repeating element
	Element
	Attribute

See “[Source document icons](#)” on page 448 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 452
- “[Building a Target Structure](#)” on page 452
- “[Modifying the Target Structure](#)” on page 453

## 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.

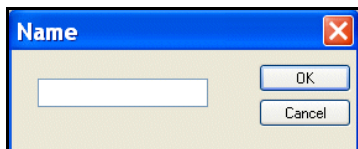
## Building a Target Structure

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 237. 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.

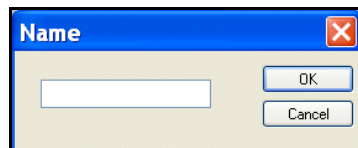


Figure 238. Name Dialog Box

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 453
- [“Removing a Node”](#) on page 454

### Adding a Node

See [“How to create elements and attributes”](#) on page 453.

### 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 456.

◆ **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 456 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 454
- [“Left and Right Mouse Buttons Explained”](#) on page 455
- [“How to Map Nodes”](#) on page 456
- [“Removing Source-Target Maps”](#) on page 456

**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 456 and [“Processing Source Nodes”](#) on page 464.

### Preserving Mapper Layout

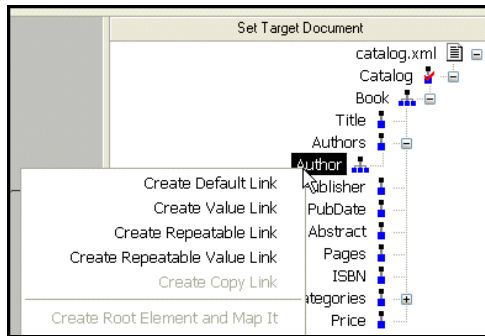
As you add function blocks to the XSLT mapper, Stylus Studio places them in the center of the mapper canvas. You can change the default placement of function blocks by dragging and drag and dropping them where you like. Stylus Studio preserves the placement you select within and across sessions (as you 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 239. 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 455.

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 `xs1:for-each` instruction. See [Working with XSLT Instructions in XSLT Mapper](#) on page 456.

### 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 442, 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 457
- [Instruction Block Ports](#) on page 458

- [Understanding Input Ports](#) on page 459
- [The Flow Port](#) on page 460
- [Adding an Instruction Block to the XSLT Mapper](#) on page 460
- [xsl:if and xsl:choose](#) on page 462

## What XSLT Instructions Are Represented Graphically

The XSLT mapper represents the following XSLT instructions:

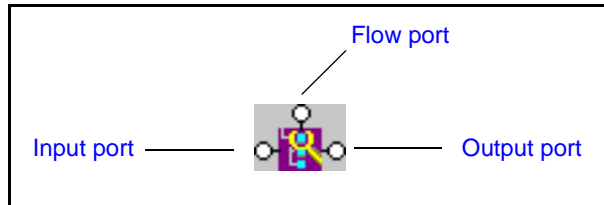
**Table 38. XSLT Instruction Blocks in XSLT Mapper**

<i>XSLT Instruction</i>	<i>Representation in the XSLT Mapper</i>
xsl:value-of	
xsl:for-each	
xsl:if	
xsl:choose	
xsl:apply-templates	
xsl:call-template	

**Note** You can create any XSLT instruction in the XSLT source, but only the instructions in [Table 38](#) 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 240](#).



**Figure 240. 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 464 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 39](#):

**Table 39. 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 select attribute. For example: <code>&lt;xsl:value-of select="'Owen' "&gt;</code>
xsl:for-each	Used to define the XPath expression for the select attribute. For example: <code>&lt;xsl:for-each select="books/book"&gt;</code>
xsl:if	Used to define the value of the test attribute. For example: <code>&lt;xsl:if test="authors/author= 'Henry' "&gt;</code> See <a href="#">xsl:if and xsl:choose</a> on page 462 to learn more about when to use this instruction.
xsl:choose	Used to define the value of the test attribute of the nested xsl:when element. For example: <code>&lt;xsl:when test="contains(authors/author, 'Marchese') "&gt;</code> See <a href="#">xsl:if and xsl:choose</a> on page 462 to learn more about when to use this instruction.
xsl:apply-templates	Used to define the value of the select attribute. For example: <code>&lt;xsl:apply-templates select="subject"/&gt;</code>
xsl:call-template	Used to define the value of the name attribute. For example: <code>&lt;xsl:call-template name="newAuthorsTemplate"/&gt;</code>

## 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 in the **Mapper** tab the next time you display it.

◆ **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 458 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 `xs1: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 `xs1:value-of` instruction is replaced by a link with a small circle at its center, link the one shown in [Figure 241](#).



**Figure 241.** `xs1:value-of` in XSLT Mapper

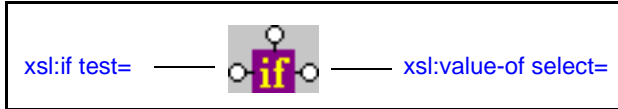
If you mouse over the circle, Stylus Studio displays the XSLT represented by the link (`xs1:value-of select="/books/book/@pubdate"/`, for example).

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 467.

- 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.

## xsl:if and xsl:choose

The `xsl:if` instruction cannot express an else condition. It has a single input port, and a single output port, as shown in [Figure 242](#).

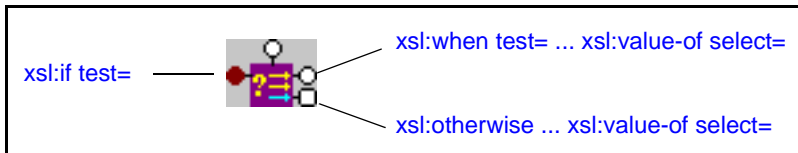


**Figure 242. xsl:if Instruction Block**

Once fully defined, the `xsl:if` block generates code like the following:

```
<Review>
  <xsl:if test="authors/author= 'Mino11o'">
    <xsl:value-of select="'Recommended'"/>
  </xsl:if>
  <xsl:if test="contains(authors/author,'Pedruzzi')">
    <xsl:value-of select="'A best buy'"/>
  </xsl:if>
</Review>
```

If you need to express an else condition, use the `xsl:choose` instruction block. This instruction block has two output ports by default, one for the `xsl:when test=` attribute, and one for the one `xsl:otherwise` element.



**Figure 243. xsl:choose Instruction Block**

The `xsl:choose` instruction block generates code like the following:

```
<Review>
  <xsl:choose>
    <xsl:when test="authors/author= 'Mino11o'">
      <xsl:value-of select="'Recommended'"/>
    </xsl:when>
    <xsl:when test="contains(authors/author,'Pedruzzi')">
      <xsl:value-of select="'authors best buy'"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="'bah...'"/>
    </xsl:otherwise>
  </xsl:choose>
</Review>
```

If you need to define more than one `xsl:when test=` attribute, use the **xsl:choose** shortcut menu (right click) and select **Add When Port**.

**Note** Stylus Studio generates the `xsl:otherwise` element by default for all `xsl:choose` instructions.

## Editing xsl:choose Instruction Properties

Use this procedure to

- Add, edit, or remove `test=` attributes from `xsl:choose` instructions.
- Edit the setting that generates the `xsl:otherwise` element.

### ◆ To edit xsl:choose properties:

1. Right click on the **xsl:choose** instruction block.
2. Select **Properties** from the shortcut menu.

The **xsl:choose** properties dialog box appears. Any attributes already defined for the `xsl:choose` instruction you are editing are displayed in the **xsl:test conditions** list box

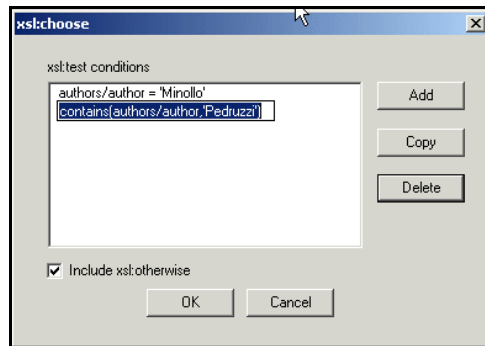


Figure 244. Properties Dialog Box for xsl:choose

3. To add a new attribute, click the **Add** button.  
A new entry field appears.
4. Replace the default value (`newCondition`) with the XPath expression you want to use to define the `test=` attribute and press Enter.

**Tip** Use the **Copy** button to create a new `test=` attribute and its nested content that is similar to an existing `test=` attribute. This feature is especially useful for attributes that are long or complex.

5. To delete an existing attribute, select it and then click the **Delete** button.
6. If you do not want the `xsl:choose` instruction to include the `xsl:otherwise` element, click the Include `xsl:otherwise` check box to deselect it.
7. When you have finished editing the `xsl:choose` properties, click **OK**.

## 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 464
- [Logical Operators](#) on page 467
- [Setting a Text Value](#) on page 467
- [Defining Java Functions in the XSLT Mapper](#) on page 469

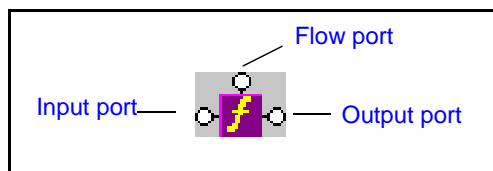
## XPath Function Blocks

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 464
- [“Types of Function Blocks”](#) on page 465
- [“Creating a Function Block”](#) on page 466
- [“Deleting a Function Block”](#) on page 466

### 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 245. 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 246. 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 247. 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

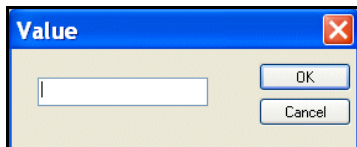
◆ **To set a text value on the mapper canvas:**

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 248. Text Block**

Double-click the text block to display the **Value** dialog box.



**Figure 249. 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.

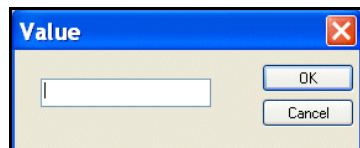


Figure 250. Value Dialog Box

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 470 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

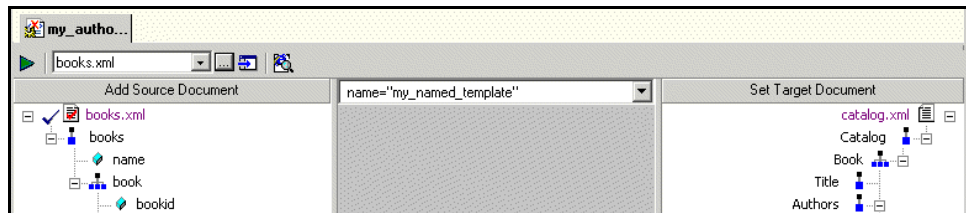
The class file must be in your CLASSPATH environment variable or in the Stylus Studio ClassPath. To add it to the Stylus Studio ClassPath, select **Tools > Options** from the Stylus Studio menu bar. In the **Options** dialog box, expand **Application Settings** and click **Java Virtual Machine**.

## 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.

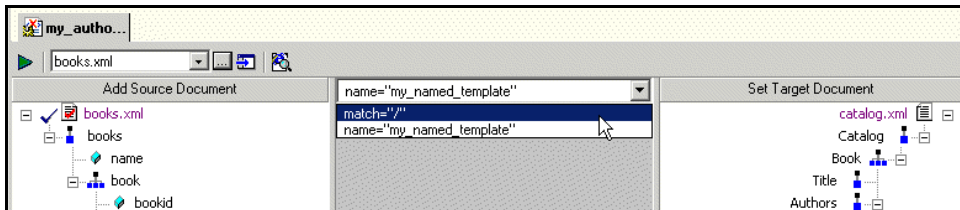
### What Happens When You Create a Template

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 251. 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 252](#).



**Figure 252. Display Different Templates Using the Drop-down List**

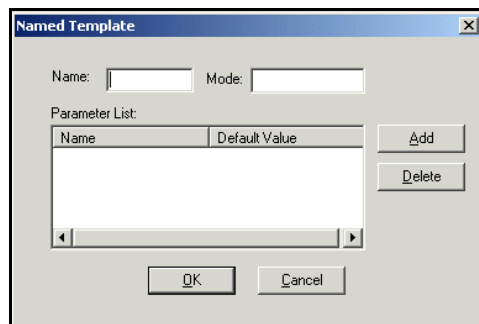
**Tip** At any time, the mapper shows only the links that have been defined for the current template.

## 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 253](#).)




**Figure 253. 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 473
- [“How to Create a Scenario”](#) on page 476
- [“How to Run a Scenario”](#) on page 477
- [“How to Clone a Scenario”](#) on page 478

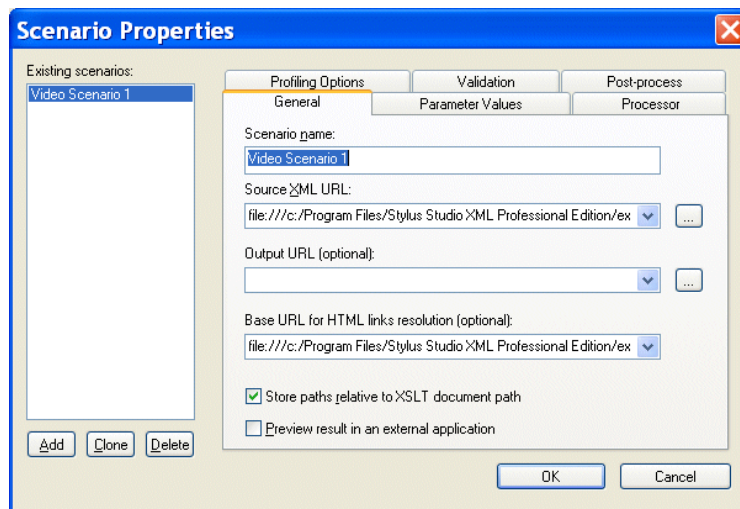
## Overview of Scenario Features

This section describes the main features of XSLT scenarios. It covers the following topics:

- “XML Source Documents” on page 473
- “Global Parameters” on page 474
- “XSLT Processors” on page 475
- “Performance Metrics Reporting” on page 475
- “Result Document Validation” on page 476
- “Post-Processing Result Documents” on page 476

### 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 254. General Tab of XSLT Scenario Properties Dialog Box**

See “Source Documents” on page 445 to learn more about the process of selecting and working with source documents in XSLT mapper.

### Global Parameters

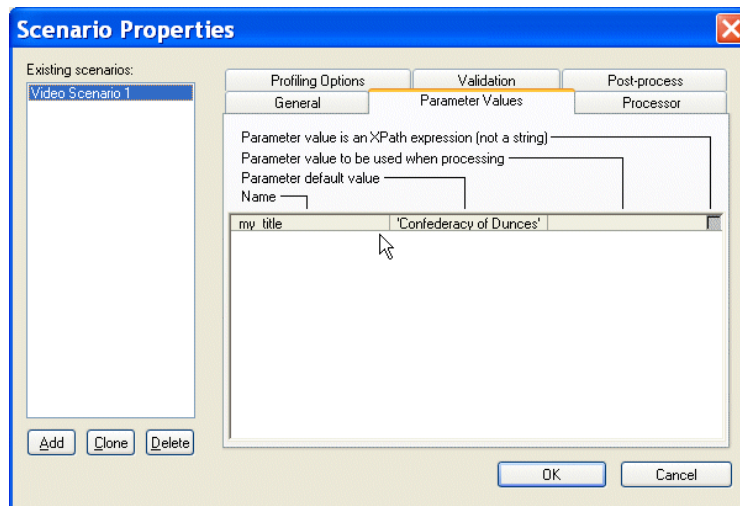
The **Parameter Values** tab of the **Scenario Properties** dialog box displays any global parameters you have defined in the XSLT source and allows you to

- 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 255. Specifying Alternate Values for XSLT Parameters**

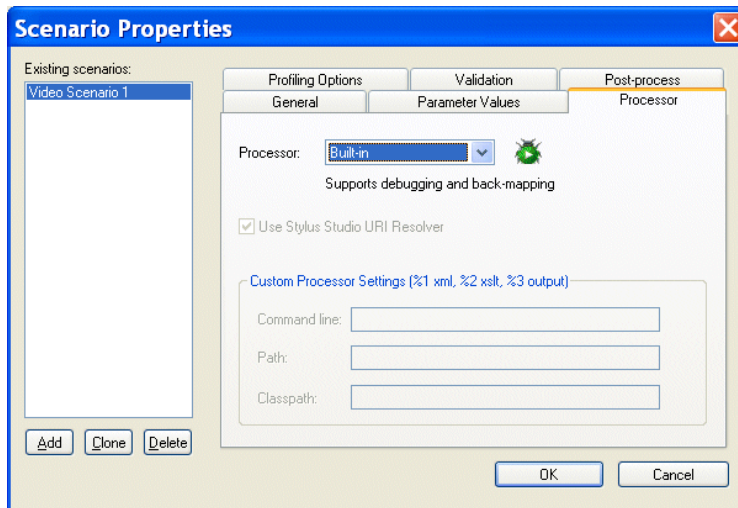
If you want to specify an alternate parameter value for this scenario, click the **Parameter value to be used when processing** entry field. If the alternate value you enter is an XPath expression, click the associated check box.

**Tip** All global parameters you define for a stylesheet are displayed on the **Params/Other** tab of the XSLT editor. Parameters displayed on this 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**.



**Figure 256. 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.

## Performance Metrics Reporting



Performance metrics reporting is available only in Stylus Studio XML Enterprise Edition.

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 486 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, Xerces-J, and XSV.

All validation is done before any post-processing that you might have specified.



See “[Validating Result Documents](#)” on page 389.

### 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 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 473 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 478.
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 476.
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, Apache Xalan-J, 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 480
- [“Using Breakpoints”](#) on page 480
- [“Viewing Processing Information”](#) on page 481
- [“Using Bookmarks”](#) on page 484
- [“Determining Which Template Generated Particular Output”](#) on page 485
- [“Determining the Output Generated by a Particular Template”](#) on page 486
- [“Profiling XSLT Stylesheets”](#) on page 486
- [“Handling Parser and Processor Errors”](#) on page 489
- [“Debugging Java Files”](#) on page 489

# 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 357.
3. [Set one or more breakpoints](#). See “[Using Breakpoints](#)” on page 480.
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 481) 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 482
- [Evaluating XPath Expressions in the Current Processor Context](#) on page 482
- [Obtaining Information About Local Variables](#) on page 482

- [Determining the Current Context in the Source Document](#) on page 483
- [Displaying a List of Process Suspension Points](#) on page 483
- [Displaying XSLT Instructions for Particular Output](#) on page 484

### 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.

### 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** .

## 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.

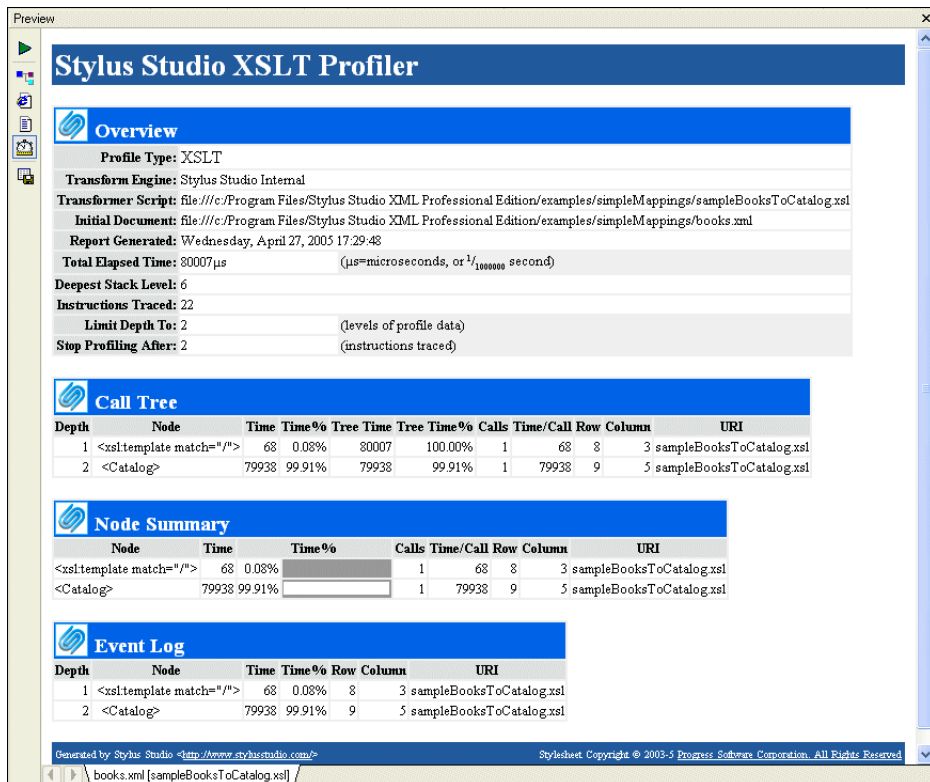
## Profiling XSLT Stylesheets



The XSLT Profiler is available only in Stylus Studio XML Enterprise Edition.

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 257](#):



**Figure 257. 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 488 for more information about this procedure.

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

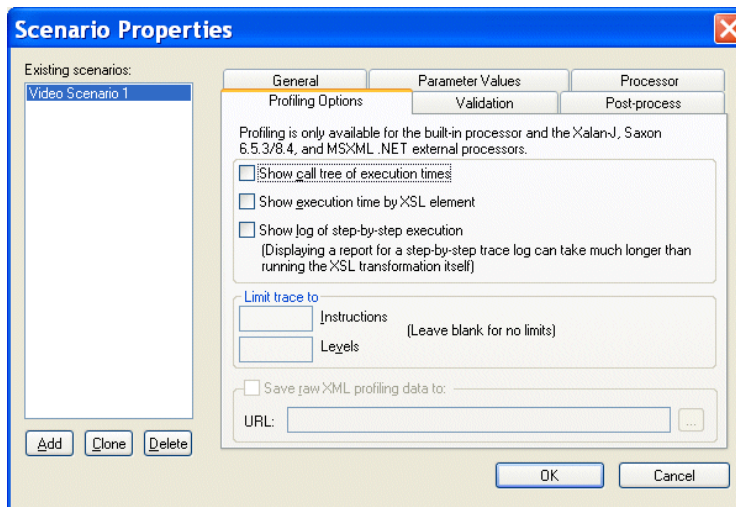
**Note** Displaying the report for a step-by-step log can take significantly longer than evaluating the XSLT itself. Consider using the Profiler with the first two performance metric options. You can also use the **Limit Trace To** fields to further restrict the Profiler's scope. If you find you need 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 258. 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 488 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 Edition and Stylus Studio XML Professional Edition.

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 490
- [Setting Options for Debugging Java](#) on page 491
- [Using the Java Editor](#) on page 491
- [Stylus Studio and JVM](#) on page 492
- [Example of Debugging Java Files](#) on page 493

## 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.

### ◆ 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 -Xrunjdpw: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 **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 138 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 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, expand **Application Settings** and click **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 479.

This section includes the following topics:


- [Setting Up to Debug Sample Java/XSLT Application](#) on page 493
- [Inserting a Breakpoint in the Sample Java/XSLT Application](#) on page 494
- [Gathering Debug Information About the Sample Java/XSLT Application](#) on page 494

## 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, expand **Application Settings** and 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. For information about the relationship between the CLASSPATH environment variable and this field, see “[About Java Virtual Machine Options](#)” on page 139.

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 498
- “Creating an XML Schema in Stylus Studio” on page 498
- “Working with XML Schema in Stylus Studio” on page 507
- “Getting Started with XML Schema in the Tree View” on page 514
- “Defining simpleTypes in XML Schemas” on page 520
- “Defining complexTypes in XML Schemas” on page 530
- “Defining Elements and Attributes in XML Schemas” on page 540
- “Defining Groups of Elements and Attributes in XML Schemas” on page 549
- “Adding Comments, Annotation, and Documentation Nodes to XML Schemas” on page 553
- “Defining Notations” on page 556
- “Referencing External XML Schemas” on page 557
- “Generating Documentation for XML Schema” on page 564
- “Generating JAXB Classes” on page 571
- “About XML Schema Properties” on page 573

# 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 499
- “[Creating XML Schema from a DTD](#)” on page 499



- “Creating XML Schema from an XML Document” on page 504
- “Creating XML Schema from EDIFACT Messages” on page 506

## 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`

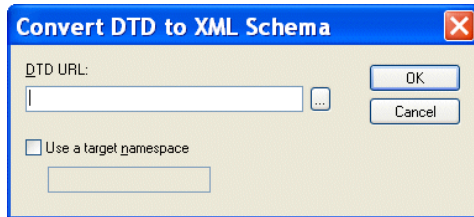
## Creating XML Schema from a DTD

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](http://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.
  2. In the **XML Editor** tab, click **DTD to XML Schema**, and click **OK**.

The **Convert DTD to XML Schema** dialog box appears.



**Figure 259. Convert DTD to XML Schema Dialog Box**

3. 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.

4. If you want, select the **Use a Target Namespace** check box and type a target namespace.
5. 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 801 for more information).

**Table 40. DTD to XML (Trang) Document Wizard Fields**

<i>Field</i>	<i>Description</i>
<b>Input file (required)</b>	The name and location of the DTD file you want to convert to XSD. You can type the file name or use the <b>Browse</b> button to browse a file system for the source DTD file.
<code>[input] xmlns=&lt;uri&gt;</code>	The default namespace; used for unqualified element names.
<code>[input] xmlns:&lt;prefix=uri&gt;</code>	The namespace for the element and autoboot names using <i>prefix</i> .

Table 40. DTD to XML (Trang) Document Wizard Fields

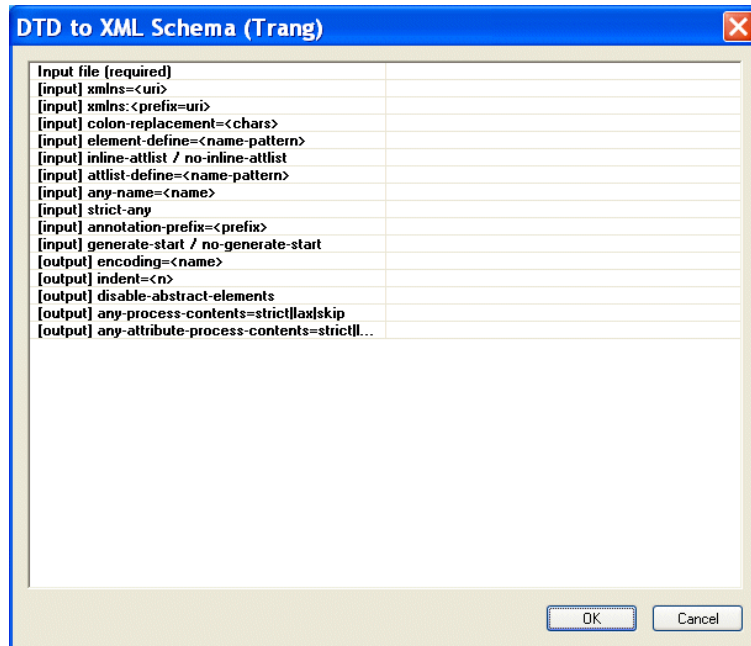
<b>Field</b>	<b>Description</b>
<b>[input] colon-replacement=&lt;chars&gt;</b>	The character that is used to replace colons in element names. Used when constructing the names of definitions used to represent the element and attribute list declarations in the DTD. Trang generates a definition for each element declaration and attlist 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.
<b>[input] element-define=&lt;name-pattern&gt;</b>	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.
<b>[input] inline-attlist / no-inline-attlist</b>	<i>inline-attlist</i> 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. <i>no-inline-attlist</i> generates a distinct definition (with <i>combine="interleave"</i> ) for each attribute list declaration in the DTD; each element declaration definition references the definition for the corresponding attribute list declaration.
<b>[input] attlist-define=&lt;name-pattern&gt;</b>	Specifies how to construct the name of the definition representing an attribute list declaration from the name of the element. The <i>name-pattern</i> 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.

**Table 40. DTD to XML (Trang) Document Wizard Fields**

<b>Field</b>	<b>Description</b>
<b>[input] any-name=&lt;name&gt;</b>	Specifies the name of the definition generated for the content of elements declared in the DTD as having a content model of ANY.
<b>[input] strict-any</b>	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.
<b>[input] annotation-prefix=&lt;prefix&gt;</b>	Default values are represented using an annotation attribute <i>prefix:defaultValue</i> where <i>prefix</i> is bound to <a href="http://relaxng.org/ns/compatibility/annotations/1.0">http://relaxng.org/ns/compatibility/annotations/1.0</a> 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.
<b>[input] generate-start / no-generate-start</b>	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.
<b>[output] encoding=&lt;name&gt;</b>	Uses <i>name</i> as the encoding for output files.
<b>[output] indent=&lt;n&gt;</b>	Indents each indent level in the output file by <i>n</i> spaces.
<b>[output] disable-abstract-elements</b>	Disables the use of abstract elements and substitution groups in the generated XML schema.
<b>[output] any-process-contents=strict lax skip</b>	Specifies the value for the processContents attribute of any elements. The default is strict, which corresponds to DTD semantics.
<b>[output] any-attribute-process-contents=strict lax skip</b>	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 260. 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.

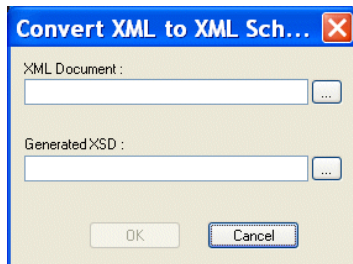
Both methods allow you to specify the URI for the generated files (if an XML document has multiple namespaces defined for it, Stylus Studio creates a separate XML Schema associated with each namespace).

### 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 261. Convert XML to XML Schema Dialog Box**

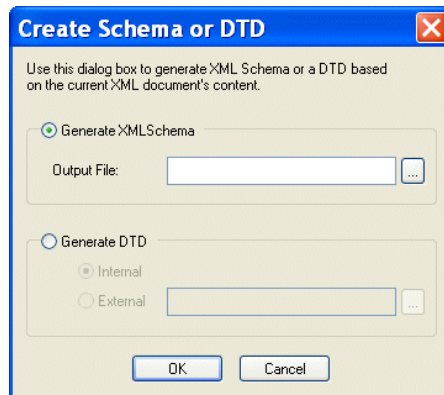
3. Specify the XML document you want to use to create an XML Schema in the **XML Document** field.
4. Specify the URI for the generated file(s) in the **Generated XSD** field.
5. Click the **OK** button.  
Stylus Studio creates the XML Schema file and opens it in the XML Schema Editor.

## 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 262. 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 EDIFACT Messages



The EDIFACT to XML Schema document wizard is available only in Stylus Studio XML Enterprise Edition.

Stylus Studio Enterprise Edition provides a document wizard that allows you to create XML Schema based on an EDIFACT message, such as the Business Credit Report (BUSCRD) message type.

### Setting Wizard Options

In addition to selecting a specific EDIFACT version and message type, Stylus Studio provides the following settings that determine how the XML Schema is generated. You can decide whether or not you want to

- Include annotations that describe each element.
- Generate enumerations for fields in the message that have lists of values
- Substitute “unbounded” for maxOccurs that have a value of 99 or more. Stylus Studio bases maxOccurs values on the loops it detects in the EDIFACT message.

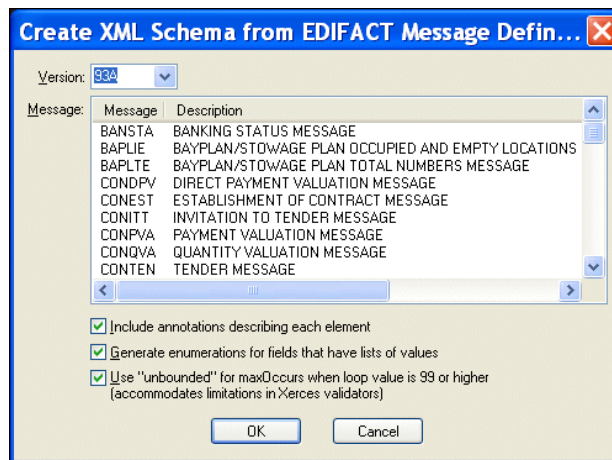
### Running the EDIFACT to XML Schema Documentation Wizard

◆ **To run the EDIFACT to XML Schema documentation wizard:**

1. Select **File > Documentation Wizards** from the Stylus Studio menu.  
The **Documentation Wizards** dialog box appears.
2. Double-click the **EDIFACT to XML Schema** icon.



The **Create XML Schema from EDIFACT Message Definition** dialog box appears.



**Figure 263. Create XML Schema from EDIFACT Message Definition**

3. If necessary, change the UN/CEFACT version ID. Versions are listed chronologically in ascending order.
4. Change the XML Schema creation options as required. See [Setting Wizard Options](#) on page 506 if you need help with this step.
5. Click OK.

Stylus Studio converts the EDIFACT message you selected in [Step 3](#) 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.

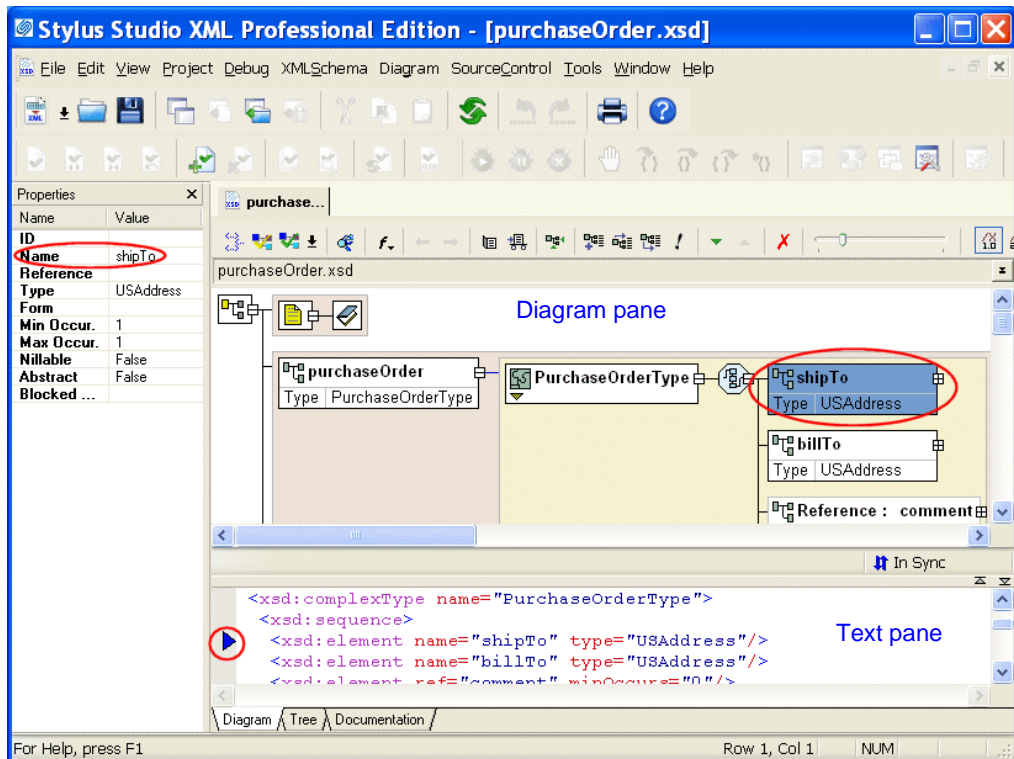
This section covers the following topics:

- [Views in the XML Schema Editor](#) on page 508
- [Validating XML Schema](#) on page 511
- [Updating XML Schema Associated with a Document](#) on page 511
- [Viewing Sample XML](#) on page 512
- [Using XML Schema in XQuery and XSLT Mapper](#) on page 513

- [Printing](#) on page 513
- [Node Properties](#) on page 513

## 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 264](#).



**Figure 264. XML Schema Editor Diagram Tab**

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 513 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.

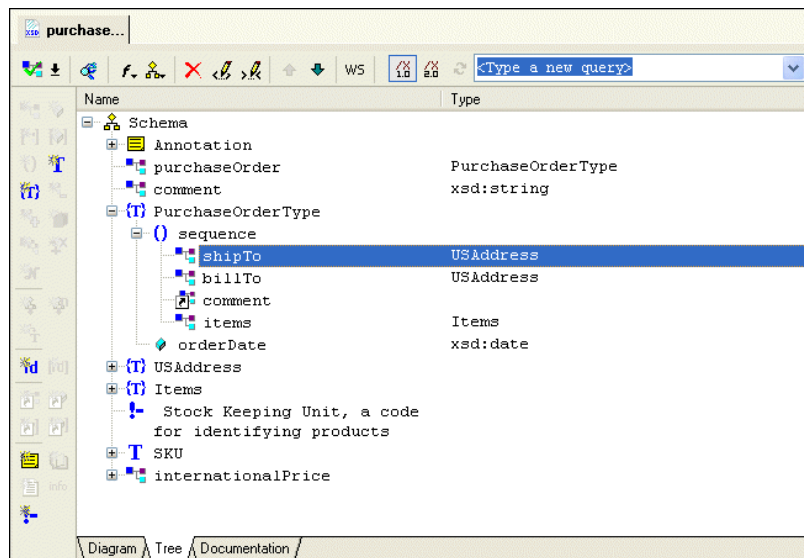
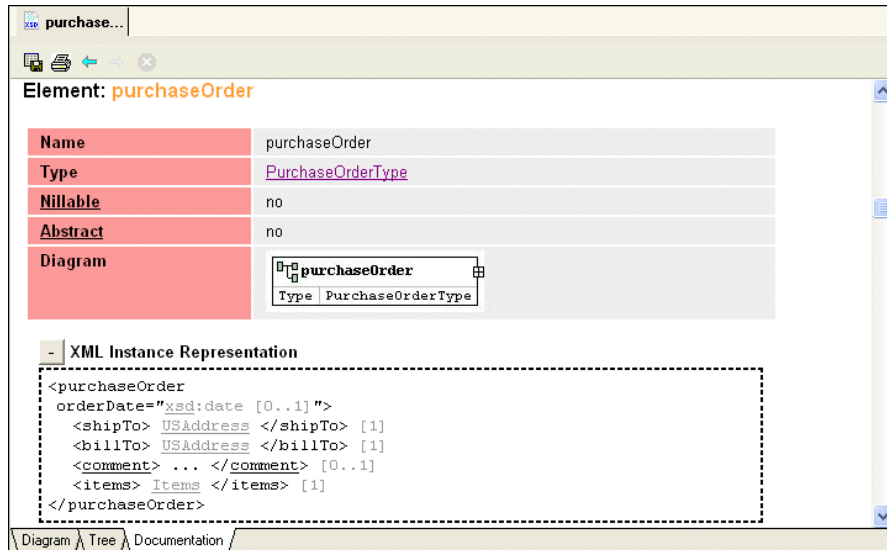


Figure 265. XML Schema Editor Tree Tab

- **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 266. 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 85. 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

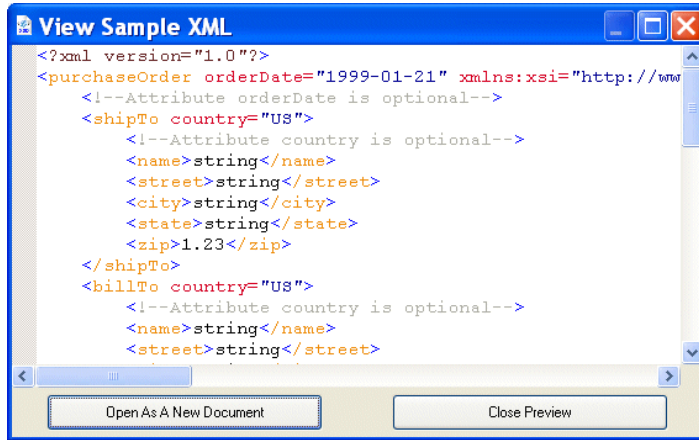
- MSXML DOM Parser
- MSXML SAX Parser
- .NET XML Parser
- Xerces-J

## 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. For example, here is a sample of the XML represented by the purchaseOrder element in purchaseOrder.xsd.



If you want, you can create a new XML document based on this fragment 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 267](#).)
3. If you want to open this sample 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 700 and [“Creating XSLT Using the XSLT Mapper”](#) on page 439 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.

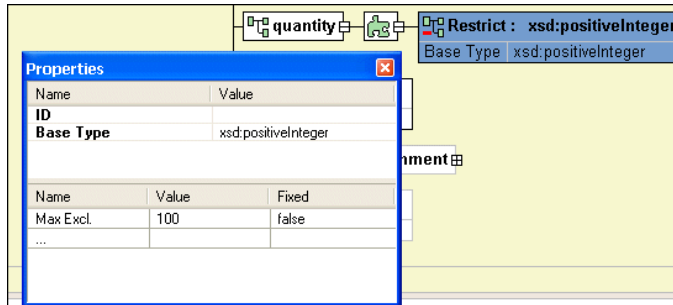
### Printing XML Schema Documentation

To print XML Schema documentation, print from the **Documentation** tab. Stylus Studio prints the XML Schema documentation using the currently selected display format (XS3P or XSDdoc). See [“Generating Documentation for XML Schema”](#) on page 564 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 268](#).



**Figure 268. 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 268](#).

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 573.

## Working with Properties in the Diagram

You can also display and edit properties within the nodes in the diagram. See “[Displaying Properties](#)” on page 89 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 515



- [Defining a complexType in a Sample XML Schema in the Tree View](#) on page 515
- [Defining Elements of the Sample complexType in the Tree View](#) on page 520

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 85.

## 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 516

- [Adding an Attribute to a Sample complexType in the Tree View](#) on page 516
- [Adding Elements to a Sample complexType in the Tree View](#) on page 517
- [Adding Optional Elements to a Sample complexType in the Tree View](#) on page 517
- [Adding an Element That Contains Subelements to a complexType in the Tree View](#) on page 518
- [Choosing the Element to Include in the Sample complexType in the Tree View](#) on page 519


### 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** .
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 521
- [Examples of simpleTypes in an XML Schema](#) on page 521
- [Defining a simpleType in the Diagram View](#) on page 522
- [Defining a simpleType in the Tree View](#) on page 527
- [About Facet Types for simpleTypes](#) on page 528

- [Defining List and Union simpleTypes in the Tree View](#) on page 530

## 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 523
- “[Defining an Atomic simpleType](#)” on page 523
- “[Specifying a Restriction for a simpleType – QuickEdit](#)” on page 523
- “[Specifying a Restriction for a simpleType – Manually](#)” on page 525
- “[Defining List and Union simpleTypes](#)” on page 526




## 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 85. 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 525.

### ◆ 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.

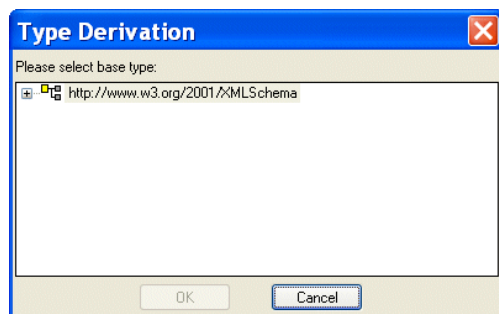


Figure 269. Type Derivation Dialog Box

The **Type Derivation** dialog box displays the W3C XML Schema, as well as any referenced XML Schemas.

- Expand the schema (click the plus sign) to display the base types associated with that XML Schema.
- Select the type on which you wish to base the simpleType you are defining and click OK.

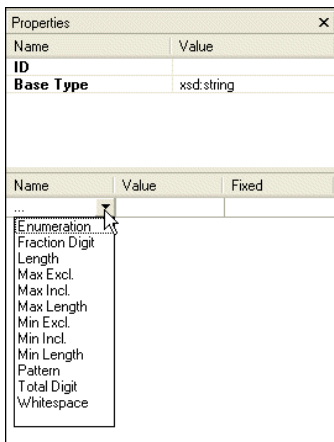
The simpleType is updated with an element that identifies the restricted type:



**Figure 270. 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.

- Click the **Name** field and select a facet type.



**Figure 271. 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 528.

- Click the **Value** field for a facet you want to specify.
- Enter a value for the new facet.
- To specify another facet, repeat [Step 5](#) through [Step 7](#).

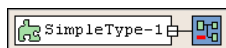
## Specifying a Restriction for a simpleType – Manually

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 523.

◆ **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:

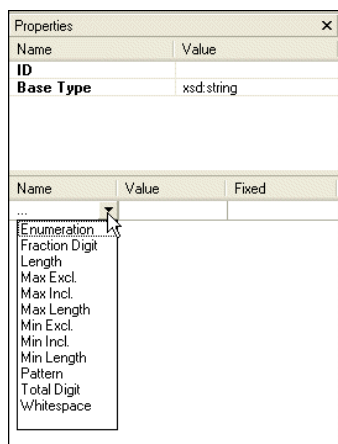


**Figure 272. Restriction Icon**

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 273. 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 528.

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](#).

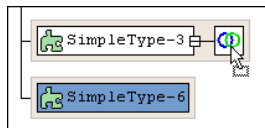
### Defining List and Union simpleTypes

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 522.
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 274](#), which shows SimpleType-6 being added to the union SimpleType-3.



**Figure 274. 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 528.

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](http://www.w3.org/TR/xmlschema-0/) 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 41](#) 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 41. 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>&lt;xsd:simpleType name="USState"&gt;   &lt;xsd:restriction base="xsd:string"&gt;     &lt;xsd:enumeration value="AK"/&gt;     &lt;xsd:enumeration value="AL"/&gt;     &lt;xsd:enumeration value="AR"/&gt;     &lt;!-- and so on ... --&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</pre>
fractionDigits	The maximum number of digits that are allowed in the fractional portion of values of simpleTypes that are derived from <code>xsd:decimal</code> .
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 <code>xsd:string</code> 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.



**Table 41. Facet Values for simpleTypes**

<i>Facet</i>	<i>Value</i>
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.
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"> <li>● preserve indicates that no normalization is done. The value is not changed.</li> <li>● replace indicates that each tab, line feed, and return is replaced with a space.</li> <li>● collapse indicates that the processing specified by replace is done, and then contiguous sequences of spaces are collapsed into one space.</li> </ul>

### 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 520.
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 544.


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 531
- [Defining complexTypes That Contain Elements and Attributes – Tree View](#) on page 535
- [Defining complexTypes That Mix Data and Elements](#) on page 537
- [Defining complexTypes That Contain Only Attributes](#) on page 539

## 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`.

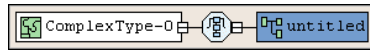
◆ **To define `xsd:choice` in the Diagram tab:**

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 275. 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 .  
Stylus Studio displays the `all` 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 `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 580.

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 531.

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 580. 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:

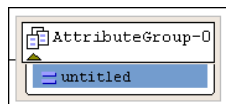
1. Right-click the schema node .
2. In the shortcut menu, select **Add > AttributeGroup**.

*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 276. 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 531.
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

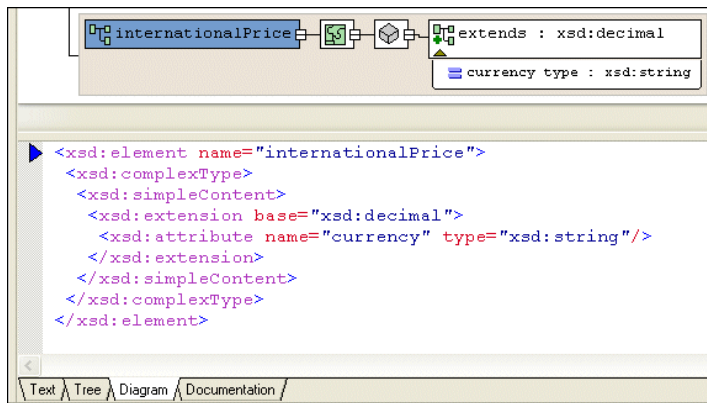
- Contain a reference to another element or attribute
- Specify values for the `minOccurs`, `maxOccurs`, or use properties

This section covers the following topics:

- [Defining Elements That Carry Attributes and Contain Data in XML Schemas](#) on page 541
- [Defining Elements That Contain Subelements in XML Schemas](#) on page 544
- [Adding an Identity Constraint to an Element](#) on page 545

## 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 277. 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:

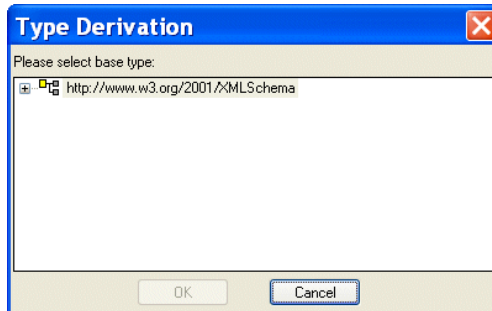
1. Right-click the schema node .
2. In the shortcut menu, select **Add > Element**.

*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 278. 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 279](#) shows an extension of the decimal simpleType.








**Figure 279. 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 531 or [“Defining complexTypes That Contain Elements and Attributes – Tree View”](#) on page 535.

### 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 535.

## 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 546
- [Diagram View](#) on page 547
- [Tree View](#) on page 548

### 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 549
- [Defining Groups of Elements in XML Schemas – Tree View](#) on page 550
- [Defining attributeGroups in XML Schemas – Diagram View](#) on page 551
- [Defining attributeGroups in XML Schemas – Tree View](#) on page 552

### 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 540.
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 540.
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 580.
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:

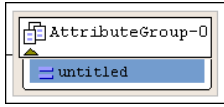
1. Right-click the schema node .
2. In the shortcut menu, select **Add > AttributeGroup**.

*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 280. 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 Edition and Stylus Studio XML Professional Edition.

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 557
- [“Where You Can Reference XML Schemas”](#) on page 558
- [“Referencing XML Schemas in the Tree View”](#) on page 561
- [“Referencing XML Schemas in the Tree View”](#) on page 561
- [“Redefining Nodes”](#) on page 561

### 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

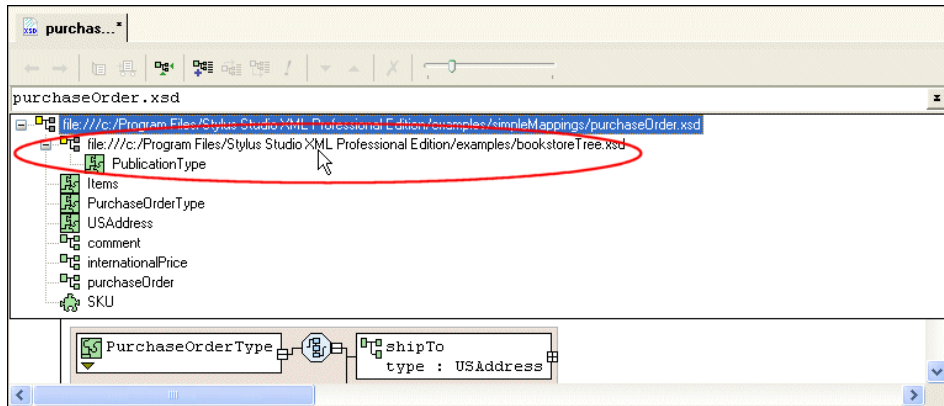
Referencing an XML Schema by *redefining* it is similar to including it, with one important difference: when you redefine an XML Schema in the base XML Schema, you can redefine the definitions of the referenced XML Schema's complexTypes, simpleTypes, groups, and attributeGroups. For example, suppose 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 281. You Can View Referenced Schemas**

If you select a referenced schema from the definition browser, as shown in [Figure 281](#), 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.

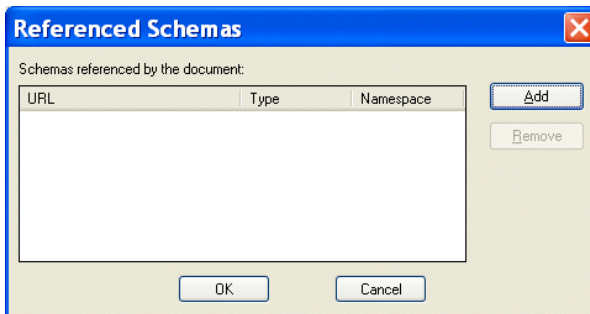
## What to Do Next

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 561 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.

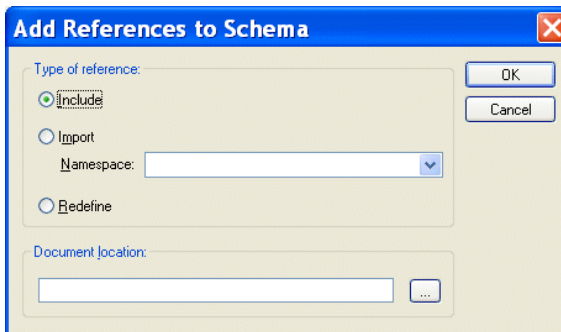
The **Referenced Schemas** dialog box appears.



**Figure 282. Referenced Schemas Dialog Box**

3. Click the **Add** button.

The **Add References to Schema** dialog box appears.






**Figure 283. 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
  - \\fileserver\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

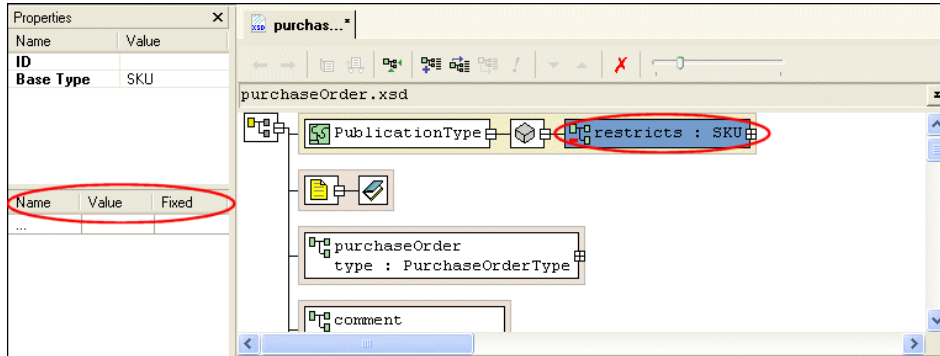
Once you reference an XML Schema by redefining it, you are able to redefine that XML Schema's complexTypes, simpleTypes, groups, and attributeGroups. This section describes how to redefine nodes using the **Diagram** tab.

## 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 284](#).




**Figure 284. 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 528 for more information on facets.

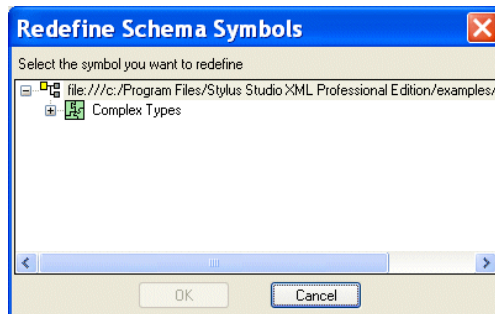
### How to Redefine a Node

◆ **To redefine a node in the Diagram view:**

1. Right-click the schema node .
2. Select **Redefine** from the shortcut menu.



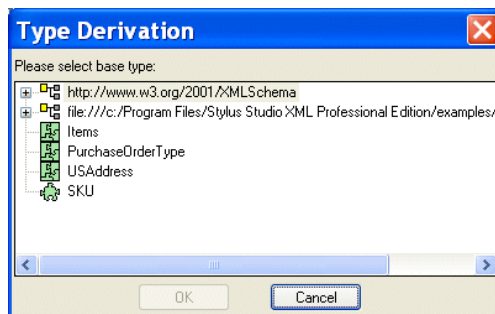
The **Redefine Schema Symbols** dialog box appears.



**Figure 285. 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**.

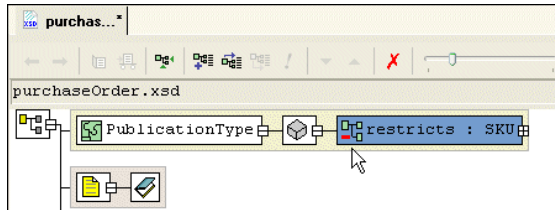
The **Type Derivation** dialog box appears.



**Figure 286. 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 287](#).



**Figure 287. 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 562 if you need help with this step.

## Generating Documentation for XML Schema



The **Documentation** tab is available only in Stylus Studio XML Enterprise Edition and Stylus Studio XML Professional Edition.

The **Documentation** tab of the XML Schema Editor displays HTML documentation that describes the currently active XML Schema. You can choose one of two display formats – XS3P (the default) or XSDdoc. Each format has its own look-and-feel, and a set of options that affects the XML Schema documentation layout and content for both display and print.

- ◆ **To display XML Schema documentation, open an XML Schema document and click the Documentation tab.**

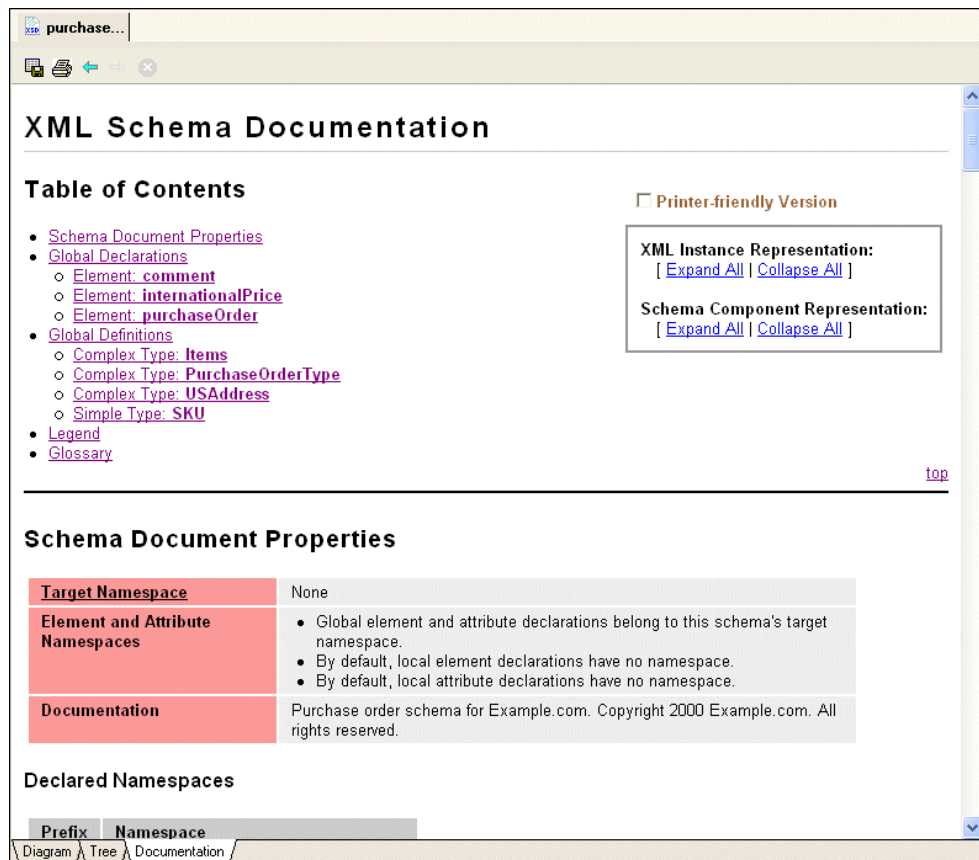
This section covers the following topics:

- [XS3P Stylesheet Overview](#) on page 565
- [XSDdoc Stylesheet Overview](#) on page 569
- [Choosing a Display Option for XML Schema Documentation](#) on page 570

- [Saving XML Schema Documentation](#) on page 571
- [Printing XML Schema Documentation](#) on page 571

## 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 288](#) shows how the `purchaseOrder.xsd` looks when displayed using this stylesheet.



**Figure 288. 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
- 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.

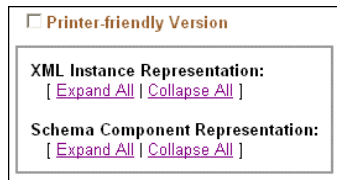
This section covers the following topics:

- [XS3P Stylesheet Features](#) on page 566
- [XS3P Stylesheet Settings](#) on page 567
- [Modifying the XS3P Stylesheet](#) on page 568

### 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.

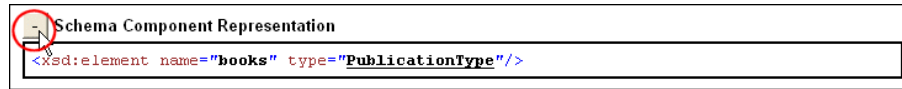


**Figure 289. Features of XML Schema Documentation**

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 290](#).



**Figure 290. 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 567 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 568.

### 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.

## XSDdoc Stylesheet Overview

The XSDdoc stylesheet, from xframe (<http://xframe.sourceforge.net/xsddoc.html/>) displays XML Schema documentation using a Javadoc look-and-feel. Here is sample of purchaseOrder.xsd displayed using the XSDdoc stylesheet.



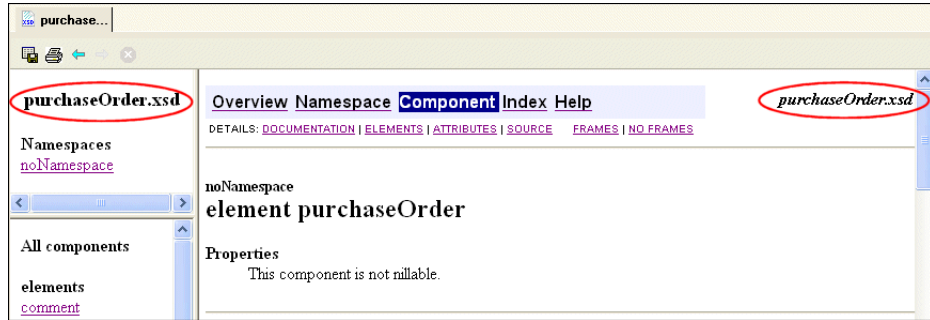
Figure 291. XML Schema Documentation Displayed Using XSDdoc Stylesheet

## XSDdoc Stylesheet Settings

The XS3P stylesheet allows you to modify the following:

- Title – The title displayed on the XML Schema documentation's *Overview* page. The default title is *XML Schema Documentation*, but you can change it to whatever you want by editing the **Doctitle** field.
- Footer – The string that appears at the bottom of every printed page in the XML Schema documentation. The default is <http://www.stylusstudio.com>.

- Header – The string that appears in the navigation bar, which is displayed at the top and bottom of every page in the XML Schema documentation. Here, the header is *purchaseOrder.xsd*.



**Figure 292. XML Schema Documentation Headers**

- Inclusion in the XML Schema documentation of
  - Attributes
  - Groups
  - Local usage
  - Subtypes
  - Types

The name or network address of the proxy host server, and the server port to which Stylus Studio attaches. The proxy host, if needed in your environment, is used to access the Internet in order to resolve imported or included stylesheets.

You control these settings on the **Documentation** page of the **Options** dialog box.


## Choosing a Display Option for XML Schema Documentation

Stylus Studio uses the XS3P stylesheet to display XML Schema documentation.


- ◆ **To change the stylesheet used to display XML Schema documentation:**
  1. Select **Tools > Options** from the Stylus Studio menu.  
The **Options** dialog box appears.
  2. Navigate to **Module Settings > XML Schema Editor > Documentation**.
  3. Select the desired display option from the **Documentation Style** group box.



4. Click OK.


This setting is used the next time you display an XML Schema on the **Documentation** tab of the XML Schema Editor. If you are viewing an XML Schema on the **Documentation** tab, click the **Reload Document** button () for the change to take effect.

## 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.

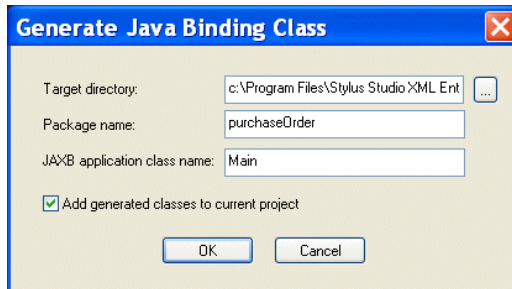
You can change any of these values at the time you run the JAXB code generator.

### 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 293. 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 Edition and Stylus Studio XML Professional Edition.

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 574
- [Element and Element Reference Properties in XML Schemas](#) on page 576
- [Attribute and Attribute Reference Properties in XML Schemas](#) on page 578
- [Group Properties in XML Schemas](#) on page 580
- [Model Group Properties in XML Schemas](#) on page 580
- [Complex and simpleType Properties in XML Schemas](#) on page 582
- [Restriction and Extension Type Properties in XML Schemas](#) on page 583

- [Content Type Properties in XML Schemas](#) on page 583
- [Aggregator Type Properties in XML Schemas](#) on page 584
- [Facet Type Properties in XML Schemas](#) on page 585
- [Notation Type Properties in XML Schemas](#) on page 586
- [Include Type Properties in XML Schemas](#) on page 586
- [Import Type Properties in XML Schemas](#) on page 587
- [Redefine Type Properties in XML Schemas](#) on page 587
- [Identity Constraint Type Properties in XML Schemas](#) on page 587
- [Constraint Element Type Properties in XML Schemas](#) on page 588
- [Documentation Type Properties in XML Schemas](#) on page 588

### 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 42](#). Click the **Tree** tab, and then click the **Schema** node to view the properties for the `xsd:schema` element.

**Table 42. `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	<p>This is the namespace that elements and attributes defined in an instance document belong to. For example, suppose you define the following:</p> <pre>&lt;xsd:schema ... targetNamespace="http://myNS"&gt;   &lt;xsd:element name="myelement"/&gt; &lt;/xsd:schema&gt;</pre> <p>In an instance document, the following declarations conform with the target namespace:</p> <pre>&lt;myelement xmlns="http://myNS"/&gt; &lt;myns:myelement xmlns:myns="http://myNS"/&gt;</pre> <p>However, the following declaration does not conform:</p> <pre>&lt;myns:myelement xmlns:myns="http://anotherNS"/&gt;</pre>
Version	Use this property as a convenient way to track the revisions of your XML Schema document.

Table 42. xsd:schema Properties

<b>Property</b>	<b>Description</b>
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 <code>form</code> attribute. If it is not explicitly defined, the schema processor checks the default attribute form specified for the Schema node. For example:</p> <pre>&lt;xs:schema elementFormDefault="qualified"   targetNamespace="http://myNs"   xmlns:myns="http://myNS"&gt;   &lt;xs:element name="topElem"&gt;     . . . .   &lt;/xs:element&gt;   &lt;xs:element name="anElem"&gt;     &lt;xs:complexType&gt;     &lt;xs:sequence&gt;     &lt;xs:element ref="myns:topElem"&gt;     . . .</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	<p>If an element does not have its own <code>blocked</code> or <code>final</code> definition, the schema processor uses the default <code>blocked</code> or <code>final</code> definition you specify here.</p>
Default Final Definitions	

## Element and Element Reference Properties in XML Schemas

Both global and local elements have the properties described in [Table 43](#). References to elements have the same properties except where noted.

**Table 43. 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 43. 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 43. 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 44](#). References to attributes have the same properties, except where noted.

**Table 44. 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 44. 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.</p> <p>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 <b>Fixed Value</b> property or a value for the <b>Default</b> 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.</p> <p>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 45](#):

**Table 45. 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 46](#):

**Table 46. 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"> <li>● <b>all</b> 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.</li> <li>● <b>choice</b> 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.</li> <li>● <b>sequence</b> 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.</li> </ul> <p>Another value you can specify is <b>any</b>. 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 47](#). simpleTypes have only the Type and Name properties.

**Table 47. 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 48](#):

**Table 48. 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 49](#):

**Table 49. 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 50](#):

**Table 50. 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"><li>● <code>list</code> indicates that all instances in the sequence must be of the same type.</li><li>● <code>union</code> indicates that the instances in the sequence can be of different types.</li></ul>
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 51](#):

**Table 51. 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>&lt;xsd:simpleType name="zip"&gt;   &lt;xsd:restriction base="xsd:string"&gt;     &lt;xsd:length value="5" fixed="true"/&gt;   &lt;/xsd:restriction&gt; &lt;/xsd:simpleType&gt;</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 <a href="#">“About Facet Types for simpleTypes”</a> on page 528.

### Notation Type Properties in XML Schemas

Notation types have the properties described in [Table 52](#). See “[Defining Notations](#)” on page 556 for more information.

**Table 52. 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, <a href="http://www.mycompany.com/mylogo.jpg">http://www.mycompany.com/mylogo.jpg</a> .

### Include Type Properties in XML Schemas

Include types have the properties described in [Table 53](#). See “[Referencing External XML Schemas](#)” on page 557 for more information.

**Table 53. 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 54](#). See “[Referencing External XML Schemas](#)” on page 557 for more information.

**Table 54. 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 55](#). See “[Referencing External XML Schemas](#)” on page 557 for more information.

**Table 55. 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 56](#). See “[Adding an Identity Constraint to an Element](#)” on page 545 for more information.

**Table 56. 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 57](#). See [“Adding an Identity Constraint to an Element”](#) on page 545 for more information.

**Table 57. 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 58. 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 589
- [“Creating DTDs”](#) on page 590
- [“About Editing DTDs”](#) on page 591
- [“About Modifiers in Element Definitions in DTDs”](#) on page 591
- [“Defining Elements in DTDs”](#) on page 595
- [“Defining General Entities and Parameter Entities in DTDs”](#) on page 604
- [“Inserting White Space in DTDs”](#) on page 606
- [“Adding Comments to DTDs”](#) on page 607
- [“About Node Properties in DTDs”](#) on page 607
- [“Associating an XML Document with an External DTD”](#) on page 611
- [“Moving an Internal DTD to an External File”](#) on page 611

### **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 227. 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 592
- [Simple Example of Aggregating Modifiers in DTDs](#) on page 593
- [More Complex Example of Aggregating Modifiers in DTDs](#) on page 593
- [Aggregating Modifiers to Allow Any Order and Any Number in DTDs](#) on page 594

## Description of Element Modifiers in DTDs

Table 59 describes the available modifiers:

**Table 59. Element Modifiers**

<b><i>Modifier</i></b>	<b><i>Description</i></b>	<b><i>Indicator in DTD</i></b>
<b>Optional</b>	This element can appear once or not at all. (0 or 1)	Question mark (?)
<b>Zero or more</b>	This element is optional and repeatable. (0, 1, or more)	Asterisk (*)
<b>One or more</b>	This element is required and repeatable. (1 or more)	Plus sign (+)
<b>Choice</b>	Exactly one of the specified subelements must appear.	Vertical bar ( )
<b>Sequence</b>	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 `paragraph`s

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

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>.

In the Tree  
tab


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 596, 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 596
  - [Specifying That an Element is Required in DTDs](#) on page 597
  - [Specifying That an Element is Optional in DTDs](#) on page 598
  - [Specifying That Multiple Instances of An Element Are Allowed in DTDs](#) on page 599
  - [Specifying That An Element Can Contain One of a Group of Elements in DTDs](#) on page 601
  - [Specifying That an Element Can Contain One or More Elements in DTDs](#) on page 602
  - [Specifying That an Element Can Contain Data in DTDs](#) on page 603
  - [Moving, Renaming, and Deleting Elements in DTDs](#) on page 603

### 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 60. Element Properties**


<i>Property</i>	<i>Value</i>
<b>Type</b>	Element
<b>Name</b>	<code>title</code>
<b>Content Model</b>	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 607.

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 596.
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.

Example 1

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
```

Example 2

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 599.

### ◆ 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 596.
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 596.
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

## Defining Document Type Definitions

---

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.

Example 1 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
```

Example 2 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 595.
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.

Example



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 595.
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

Specifying  
Modifiers



that can appear one or more times, you must perform steps 1 through 4 for each element.

Modifying  
the Order

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 596.

◆ **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.

## Defining Document Type Definitions

---

Rename	To rename an element or attribute, right-click it and select <b>Rename</b> from the shortcut menu that appears. Type the new name and press Enter. <i>Alternative:</i> Click <b>Change Name</b>  .
Delete	To delete a node in the DTD, right-click the node you want to delete. In the shortcut menu that appears, click <b>Delete</b> . <i>Alternative:</i> Click <b>Delete Node</b>  .

## 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 610.

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 605
- [General Entity Example in a DTD](#) on page 606
- [Parameter Entity Example in a DTD](#) on page 606

## 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 608
- [Description of Attribute Properties in DTDs](#) on page 608
- [Description of Entity and Parameter Entity Properties in DTDs](#) on page 610

### 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 61](#) describes the possible values of the Content Model property for Element nodes:

**Table 61. 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 62](#) shows the properties that an attribute can have. It also provides the possible values, and a description for each property.

**Table 62. 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.

**Table 62. Attribute Property Descriptions**

<i>Property</i>	<i>Allowable Values</i>	<i>Description</i>
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.
	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 62. 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 63 shows the properties that an entity or parameter entity can have. It also provides the possible values, and a description for each property.

**Table 63. 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 63. 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 XPath processor allows you to retrieve a subset of an XML document. A query, which is always an XPath expression, returns a well-formed XML node-list or an XPath value object. In XPath 2.0, an expression returns a sequence of XML nodes and/or XPath value objects.

This section discusses the following topics:

- [“About the XPath Processor”](#) on page 614
- [“Sample Data for Examples and Testing”](#) on page 616
- [“Getting Started with Queries”](#) on page 621
- [“Specifying the Nodes to Evaluate”](#) on page 634
- [“Handling Strings and Text”](#) on page 648
- [“Specifying Boolean Expressions and Functions”](#) on page 655
- [“Specifying Number Operations and Functions”](#) on page 658
- [“Comparing Values”](#) on page 661
- [“Finding a Particular Node”](#) on page 666
- [“Obtaining a Union”](#) on page 674
- [“Obtaining Information About a Node or a Node Set”](#) on page 675
- [“Using XPath Expressions in Stylesheets”](#) on page 679
- [“Accessing Other Documents During Query Execution”](#) on page 683
- [“XPath Quick Reference”](#) on page 685

# About the XPath Processor

The XML editor in Stylus Studio supports both the W3C XPath 1.0 Recommendation and the W3C XPath 2.0 Working Draft of November 2003. By default, Stylus Studio uses its XPath 1.0 processing engine. You can select the XPath 2.0 processing engine by clicking the **v.2** icon (  ) in the XML editor tool bar.

As an overview of the XPath processor, this section provides the following information:

- [Where You Can Specify Queries](#) on page 614
- [About XPath](#) on page 614
- [Benefits of XPath](#) on page 615
- [Internationalization](#) on page 616
- [Restrictions on Queries](#) on page 616

For additional information about XPath see <http://www.w3.org/TR/xpath> and <http://www.w3.org/TR/xpath20>.

## Where You Can Specify Queries

When you use Stylus Studio, you specify queries as values of `match` and `select` attributes in stylesheets. In this way, you use queries to select the nodes you want to transform and query.

You can also specify a query at the top of an XML document window. 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 618.

### 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.

## Sample Data for Examples and Testing

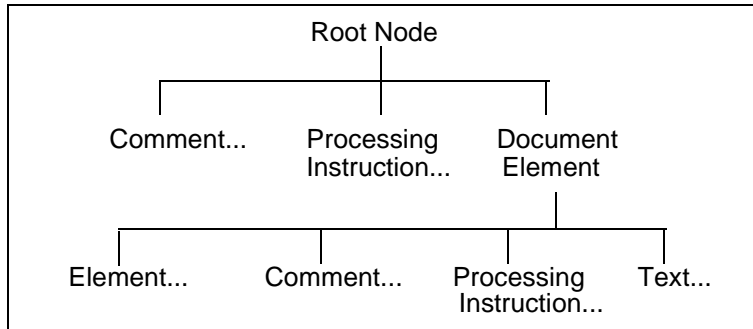
The best way to learn how to query data is to practice using queries. To prepare you for testing queries, this section provides a review of the basic structure of an XML document. An understanding of this structure is crucial to defining queries that return the data you want. Following the review, this 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 617
- [“A Sample XML Document”](#) on page 618
- [“Tree Representation of a Sample XML Document”](#) on page 618
- [“Steps for Trying the Sample Queries”](#) on page 620

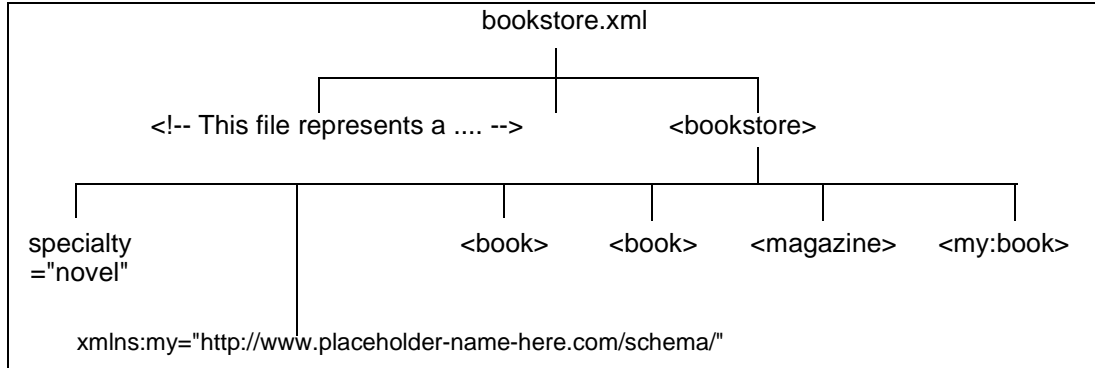
## About XML Document Structure

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 618

### 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 294](#) (and is



continued in [Figure 295](#)). 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 294. Tree Display of an XML Document

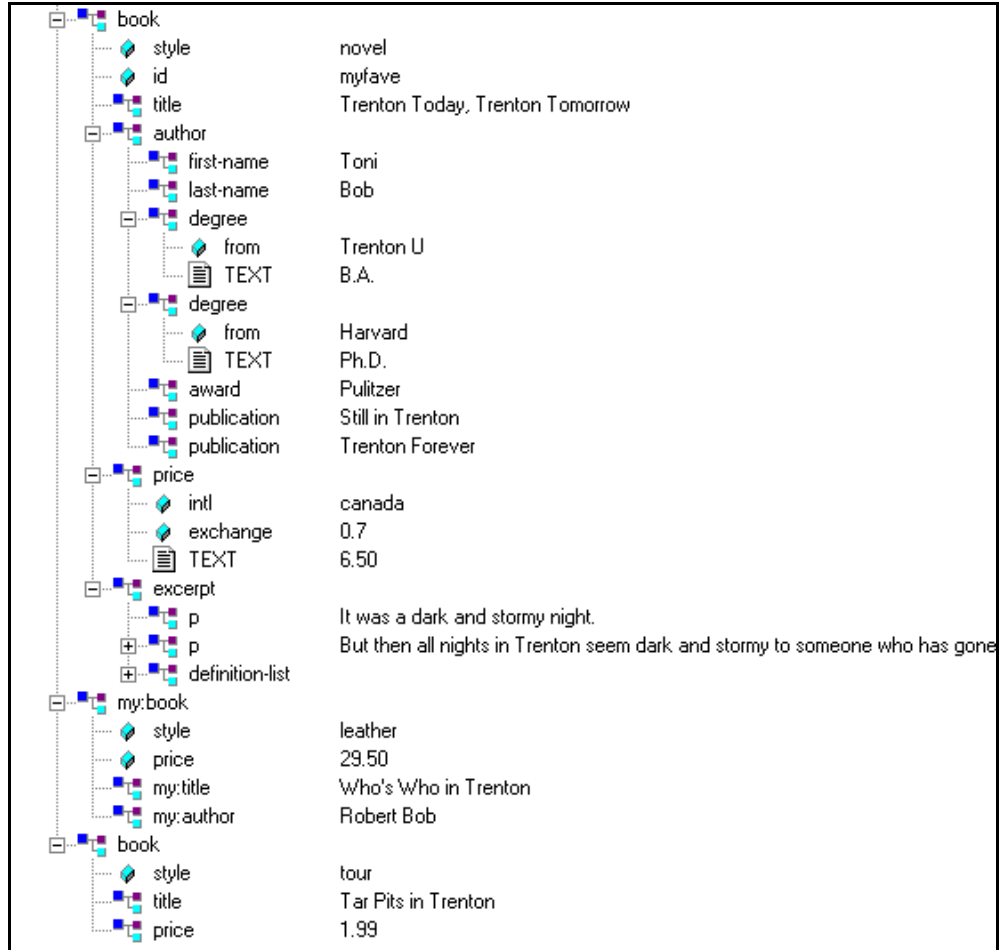



Figure 295. Tree Display of an XML Document (continued)

## 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. At the top of the window that contains bookstore.xml, highlight **<Type a new query>**.

3. Type a query. For example: `/bookstore/book/author`.
  4. Press Enter or click **Refresh 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.

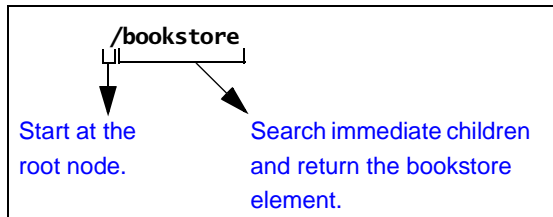
The topics discussed in this section include

- [“Obtaining All Marked-Up Text”](#) on page 621
- [“Obtaining a Portion of an XML Document”](#) on page 622
- [“Obtaining All Elements of a Particular Name”](#) on page 623
- [“Obtaining All Elements of a Particular Name from a Particular Branch”](#) on page 624
- [“Different Results from Similar Queries”](#) on page 625
- [“Queries That Return More Than You Want”](#) on page 625
- [“Specifying Attributes in Queries”](#) on page 626
- [“Filtering Results of Queries”](#) on page 627
- [“Wildcards in Queries”](#) on page 630
- [“Calling Functions in Queries”](#) on page 631
- [“Case Sensitivity and Blank Spaces in Queries”](#) on page 632
- [“Precedence of Query Operators”](#) on page 633

### 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`:

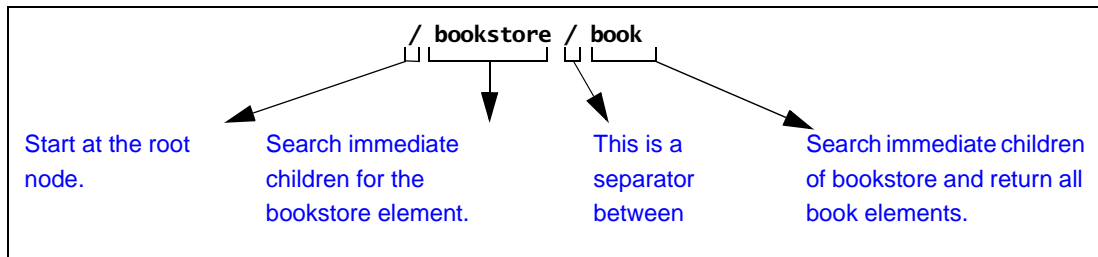
```
/book
```

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.

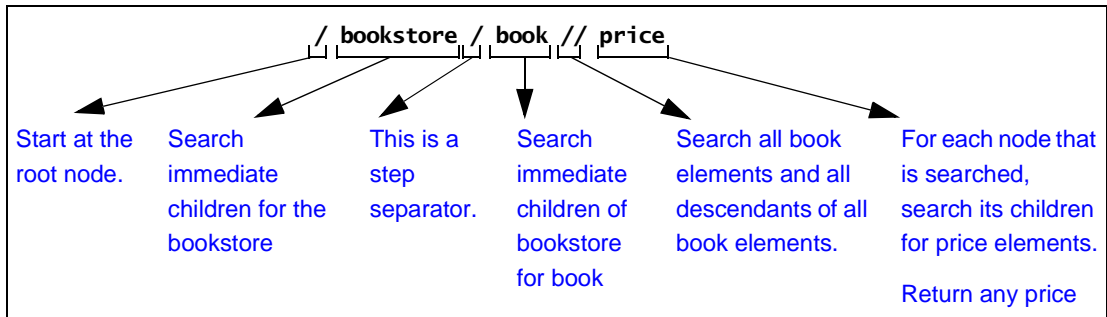
### Obtaining All Elements of a Particular Name from a Particular Branch

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:

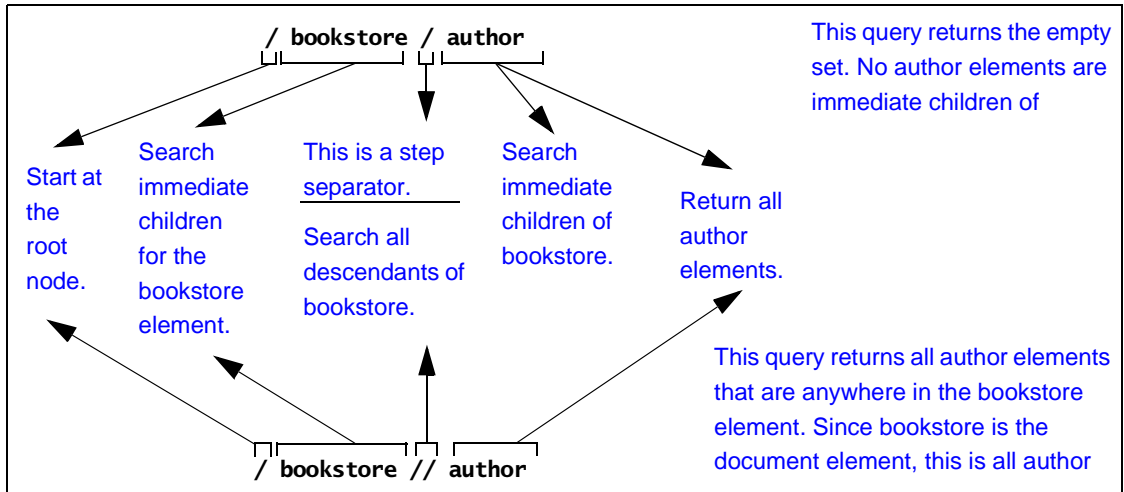
```
/bookstore/book//price
```

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

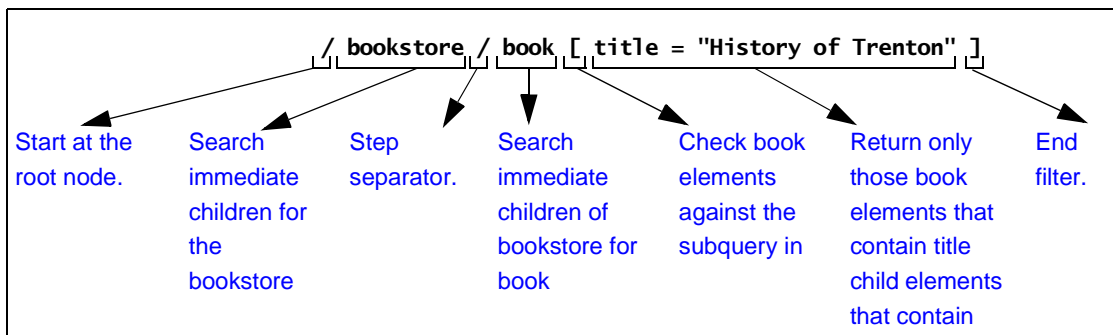
You can use an at sign (@) and asterisk (\*) together to retrieve a collection of attributes. For example, the following query finds all attributes in the current context:

```
@*
```

## 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 628
- [“More Filter Examples”](#) on page 628
- [“How the XPath Processor Evaluates a Filter”](#) on page 629

- [“Multiple Filters”](#) on page 629
- [“Filters and Attributes”](#) on page 630

### 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 686.

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 64. 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 65. 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 65. 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 635
- [“Starting at the Context Node”](#) on page 637
- [“About Root Nodes and Document Elements”](#) on page 637
- [“Starting at the Root Node”](#) on page 637
- [“Descending Along Branches”](#) on page 638
- [“Explicitly Specifying the Current Context”](#) on page 639
- [“Specifying Children or Descendants of Parent Nodes”](#) on page 640
- [“Examples of XPath Expression Results”](#) on page 640
- [“Syntax for Specifying an Axis in a Query”](#) on page 641



- [“Supported Axes”](#) on page 642
- [“Axes That Represent the Whole XML Document”](#) on page 647

## 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 641.

### 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 627). 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 66 provides examples of XPath expression results:

**Table 66. 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 66. XPath Expression Results

<b>Expression</b>	<b>Result</b>
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 691.

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 642.

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 `.` 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
```

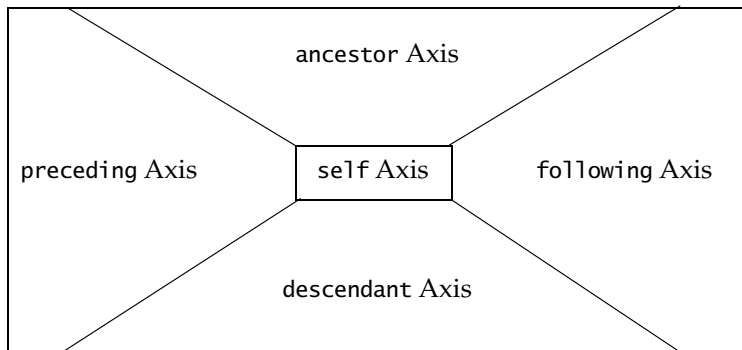
If the context node is the award element in the first book element, this query returns the first author element.

## 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 648
- [“Manipulating Strings”](#) on page 651
- [“Obtaining the Text Contained in a Node”](#) on page 654

## Searching for Strings

This section provides information about searching for strings. This section discusses the following topics:

- [“Finding Identical Strings”](#) on page 648
- [“Finding Strings That Contain Strings You Specify”](#) on page 649
- [“Finding Substrings That Appear Before Strings You Specify”](#) on page 649
- [“Finding Substrings That Appear After Strings You Specify”](#) on page 650
- [“Finding Substrings by Position”](#) on page 650

## 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 651
- [“Determining the Number of Characters in a String”](#) on page 651
- [“Normalizing Strings”](#) on page 652
- [“Replacing Characters in Strings with Characters You Specify”](#) on page 652
- [“Converting Objects to Strings”](#) on page 653
- [“Finding Strings That Start with a Particular String”](#) on page 654

### 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"><li>● NaN (not a number) becomes "NaN"</li><li>● Positive zero becomes "0"</li><li>● Negative zero becomes "0"</li><li>● Positive infinity becomes "Infinity"</li><li>● Negative infinity becomes "-Infinity"</li><li>● 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".</li><li>● 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.</li></ul>
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 653.

## 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 655
- [“Calling Boolean Functions”](#) on page 656

### 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 656
- [“Obtaining Boolean Values”](#) on page 657
- [“Determining the Context Node Language”](#) on page 657

### 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 658
- [“Calling Number Functions”](#) on page 659

### 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 659.

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 659
- [Obtaining the Sum of the Values in a Node Set](#) on page 660
- [Obtaining the Largest, Smallest, or Closest Number](#) on page 660

## 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 67](#) shows how the `number()` function converts its argument to a number:

**Table 67. `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 662
- [“How the XPath Processor Evaluates Comparisons”](#) on page 662
- [“Comparing Node Sets”](#) on page 663
- [“Comparing Single Values With = and !=”](#) on page 664
- [“Comparing Single Values With <=, <, >, and >=”](#) on page 665
- [“Priority of Object Types in Comparisons”](#) on page 665

- [“Examples of Comparisons”](#) on page 666
- [“Operating on Boolean Values”](#) on page 666

## About Comparison Operators

The comparison operators you can specify are listed in [Table 68](#):

**Table 68. 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 627, 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.

## 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 69. Comparison Operator Descriptions**

<i>Comparison</i>	<i>True If and Only If</i>
<code>&lt;</code>	The first number is less than the second number.
<code>&lt;=</code>	The first number is less than or equal to the second number.
<code>&gt;</code>	The first number is greater than the second number.
<code>&gt;=</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 667
- [“Determining the Position Number of a Node”](#) on page 667
- [“Positions in Relation to Parent Nodes”](#) on page 668
- [“Finding Nodes Relative to the Last Node in a Set”](#) on page 669
- [“Finding Multiple Nodes”](#) on page 669
- [“Examples of Specifying Positions”](#) on page 670
- [“Finding the First Node That Meets a Condition”](#) on page 670
- [“Finding an Element with a Particular ID”](#) on page 671
- [“Obtaining Particular Types of Nodes By Using Node Tests”](#) on page 672

See also [“Obtaining the Current Node for the Current XSLT Template”](#) on page 680.



## 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 667.

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 671.

## 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 667. For information about `last()`, see [“Determining the Context Size”](#) on page 678.

## 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 671
- [“Unique IDs”](#) on page 671

### 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 70](#). Node tests unique to XPath 2.0 are shown in [Table 70](#).

**Table 70. Node Test Return Values Common to XPath 1.0 and XPath 2.0**

<b><i>Node Test</i></b>	<b><i>Node Type Returned</i></b>
<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 71. Node Test Return Values Unique to XPath 2.0**

<b><i>Node Test</i></b>	<b><i>Node Type Returned</i></b>
<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 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 675
- [“Obtaining Namespace Information”](#) on page 675
- [“Obtaining the URI for an Unparsed Entity”](#) on page 678
- [“Determining the Number of Nodes in a Collection”](#) on page 678
- [“Determining the Context Size”](#) on page 678

### 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 627.

**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 627.

### Obtaining Namespace Information

You can call functions to obtain namespace information. This topic discusses

- [“Obtaining the Namespace URI”](#) on page 676
- [“Obtaining the Local Name”](#) on page 676
- [“Obtaining the Expanded Name”](#) on page 676

In addition to a discussion of the functions you call, this section covers the following:

- [“Specifying Wildcards with Namespaces”](#) on page 677
- [“Examples of Namespaces in Queries”](#) on page 677

### 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

```
string unparsed-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 679
- [“Obtaining System Properties”](#) on page 679
- [“Determining If Functions Are Available”](#) on page 680
- [“Obtaining the Current Node for the Current XSLT Template”](#) on page 680
- [“Finding an Element with a Particular Key”](#) on page 681
- [“Generating Temporary IDs for Nodes”](#) on page 683

### 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 427
- `xs1:variable` on page 436

### 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 Sonic Software 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)
```

## Writing XPath Expressions

---

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.

Example

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 684
- [“When the First Argument is a Node Set”](#) on page 684
- [“Specification of Second Argument”](#) on page 684
- [“Example of Calling the document\(\) Function”](#) on page 685

### 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 686
- [“XPath Syntax Quick Reference”](#) on page 689
- [“XPath Abbreviations Quick Reference”](#) on page 691

See also [“Precedence of Query Operators”](#) on page 633.

## XPath Functions Quick Reference

Table 72 lists the functions you can call in a query and provides short descriptions.

**Table 72. XPath Function Quick Reference**

<b>Function</b>	<b>Source</b>	<b>Returns</b>
<code>boolean()</code>	XPath	Boolean value that is the result of converting an object to a Boolean value. See <a href="#">“Converting an Object to Boolean”</a> on page 656.
<code>ceiling()</code>	XPath	Number that is the smallest integer that is not less than a number you specify. See <a href="#">“Obtaining the Largest, Smallest, or Closest Number”</a> on page 660.
<code>comment()</code>	XPath	Comment nodes. See <a href="#">“Obtaining Particular Types of Nodes By Using Node Tests”</a> on page 672.
<code>concat()</code>	XPath	String that concatenates two or more strings you specify. See <a href="#">“Concatenating Strings”</a> on page 651.
<code>contains()</code>	XPath	Nodes that contain the specified string. See <a href="#">“Searching for Strings”</a> on page 648.
<code>count()</code>	XPath	Number of nodes in the <i>node-set</i> argument. See <a href="#">“Determining the Number of Nodes in a Collection”</a> on page 678.
<code>current()</code>	XPath	Node for which the current template started its operation. See <a href="#">“Obtaining the Current Node for the Current XSLT Template”</a> on page 680.
<code>document()</code>	XSLT	Root node of the specified document. See <a href="#">“Accessing Other Documents During Query Execution”</a> on page 683.
<code>element-available()</code>	XSLT	Boolean value that indicates whether the specified element is supported by the XSLT processor. See <a href="#">“Determining If Functions Are Available”</a> on page 680.
<code>false()</code>	XPath	false. See <a href="#">“Obtaining Boolean Values”</a> on page 657.

Table 72. XPath Function Quick Reference

<b>Function</b>	<b>Source</b>	<b>Returns</b>
<code>floor()</code>	XPath	Number that is the largest integer that is not greater than a number you specify. See <a href="#">“Obtaining the Largest, Smallest, or Closest Number”</a> on page 660.
<code>function-available()</code>	XSLT	Boolean value that indicates whether the specified function is supported by the XPath processor. See <a href="#">“Determining If Functions Are Available”</a> on page 680.
<code>generate-id()</code>	XSLT	String that uniquely, temporarily, identifies a node. See <a href="#">“Generating Temporary IDs for Nodes”</a> on page 683.
<code>id()</code>	XPath	Element whose <code>id</code> attribute value matches the specified value. See <a href="#">“Finding an Element with a Particular ID”</a> on page 671.
<code>key()</code>	XSLT	Node whose key value matches the specified key. See <a href="#">“Finding an Element with a Particular Key”</a> on page 681.
<code>lang()</code>	XPath	Boolean value that indicates whether the language of the node is the language you expect. See <a href="#">“Determining the Context Node Language”</a> on page 657.
<code>last()</code>	XPath	Number of nodes in the context list. See <a href="#">“Determining the Number of Nodes in a Collection”</a> on page 678.
<code>local-name()</code>	XPath	Local portion of the node name, excluding the prefix. See <a href="#">“Obtaining Namespace Information”</a> on page 675.
<code>name()</code>	XPath	String that contains the tag name of the node, including namespace information, if any. See <a href="#">“Obtaining Namespace Information”</a> on page 675.

**Table 72. XPath Function Quick Reference**

<b>Function</b>	<b>Source</b>	<b>Returns</b>
namespace-uri()	XPath	URI for the namespace of the node. See <a href="#">“Obtaining Namespace Information”</a> on page 675.
node()	XPath	All nonattribute nodes. See <a href="#">“Obtaining Particular Types of Nodes By Using Node Tests”</a> on page 672.
normalize-space()	XPath	String without leading or trailing white space. See <a href="#">“Normalizing Strings”</a> on page 652.
not()	XPath	Boolean value that indicates the opposite of the specified Boolean value. See <a href="#">“Obtaining Boolean Values”</a> on page 657.
number()	XPath	Number that is the result of converting the specified argument to a number. See <a href="#">“Converting an Object to a Number”</a> on page 659.
position()	XPath	Position number of the node relative to the context node set. See <a href="#">“Finding a Particular Node”</a> on page 666.
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 <a href="#">“Obtaining Particular Types of Nodes By Using Node Tests”</a> on page 672.
round()	XPath	Number that is the closest to the argument and is an integer. See <a href="#">“Obtaining the Largest, Smallest, or Closest Number”</a> on page 660.
starts-with()	XPath	Boolean value that indicates if a string starts with a particular string. See <a href="#">“Finding Strings That Start with a Particular String”</a> on page 654.
string()	XPath	String that is the result of converting some object to a string. See <a href="#">“Converting Objects to Strings”</a> on page 653.

Table 72. XPath Function Quick Reference

<i>Function</i>	<i>Source</i>	<i>Returns</i>
string-length()	XPath	Number of characters in a string you specify. See <a href="#">“Determining the Number of Characters in a String”</a> on page 651.
substring()	XPath	Substring that is in a particular position within its string. See <a href="#">“Finding Substrings by Position”</a> on page 650.
substring-before()	XPath	Substring that appears before a string you specify. See <a href="#">“Finding Substrings That Appear Before Strings You Specify”</a> on page 649.
substring-after()	XPath	Substring that appears after a string you specify. See <a href="#">“Finding Substrings That Appear After Strings You Specify”</a> on page 650.
sum()	XPath	Number that is the sum of the values of the nodes in the specified set. See <a href="#">“Obtaining the Sum of the Values in a Node Set”</a> on page 660.
system-property()	XSLT	Object that represents the specified property. See <a href="#">“Obtaining System Properties”</a> on page 679.
translate()	XPath	String with some characters replaced by other characters. See <a href="#">“Replacing Characters in Strings with Characters You Specify”</a> on page 652.
true()	XPath	true. See <a href="#">“Obtaining Boolean Values”</a> on page 657.
unparsed-entity-uri()	XSLT	URI of an unparsed entity with the specified name. See <a href="#">“Obtaining the URI for an Unparsed Entity”</a> on page 678.

## 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 73 defines the abbreviations you can use in XPath expressions:

**Table 73. XPath Abbreviations Quick Reference**

<b>Abbreviation</b>	<b>Description</b>
No axis is specified in a location step.	The child axis is assumed. For example, the following two XPath expressions both return the para children of chapter children of the context node: chapter/para child::chapter/child::para
@	The attribute axis. For example, the following two XPath expressions both return para children of the context node that have type attributes with a value of warning: para[@type="warning"] child::para[attribute::type="warning"]

**Table 73. XPath Abbreviations Quick Reference**

<b>Abbreviation</b>	<b>Description</b>
//	<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 74 shows examples of abbreviations in XPath expressions

**Table 74. Abbreviations in XPath Expressions**

<b>Example</b>	<b>Description</b>
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 74. Abbreviations in XPath Expressions

<i>Example</i>	<i>Description</i>
para[ <i>last()</i> ]	Selects the last <i>para</i> child of the context node
<i>/doc/chapter</i> [5]/ <i>section</i> [2]	Selects the second <i>section</i> of the fifth <i>chapter</i> of the <i>doc</i> child of the context node
para[@ <i>type</i> ="warning"]	Selects <i>para</i> children of the context node that have <i>type</i> attributes with a value of <i>warning</i>
para[@ <i>type</i> ="warning"][5]	Selects the fifth <i>para</i> child of the context node that has a <i>type</i> attribute with a value of <i>warning</i>
para[5][@ <i>type</i> ="warning"]	Selects the fifth <i>para</i> child of the context node if that child has a <i>type</i> attribute with a value of <i>warning</i>
chapter[ <i>title</i> ]	Selects the <i>chapter</i> children of the context node that have one or more <i>title</i> children
<i>//para</i>	Selects all <i>para</i> descendants of the document root
chapter <i>//para</i>	Selects all <i>para</i> descendants of <i>chapter</i> children of the context node
<i>//olist/item</i>	Selects all <i>item</i> elements that have <i>olist</i> parents
.	Selects the context node
<i>./para</i>	Selects the <i>para</i> descendants of the context node
<i>..</i>	Selects the parent of the context node
@*	Selects all attributes of the context node
@ <i>name</i>	Selects the <i>name</i> attribute of the context node
<i>../@name</i>	Selects the <i>name</i> 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 Edition and Stylus Studio XML Professional Edition.

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 XQueries.

For more information

Stylus Studio provides video demonstrations for Stylus Studio's XQuery features. Visit our web site to view them and other Stylus Studio video demonstrations:

[http://www.StylusStudio.com/xml\\_videos.html](http://www.StylusStudio.com/xml_videos.html)

Contents

This chapter covers the following topics:

- “Getting Started with XQuery in Stylus Studio” on page 695
- “Building an XQuery Using the Mapper” on page 700
- “Debugging XQuery Documents” on page 724
- “Creating an XQuery Scenario” on page 731

### 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 696
- [The Stylus Studio XQuery Editor](#) on page 696

### 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 query, 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

For more information about XQuery, 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.

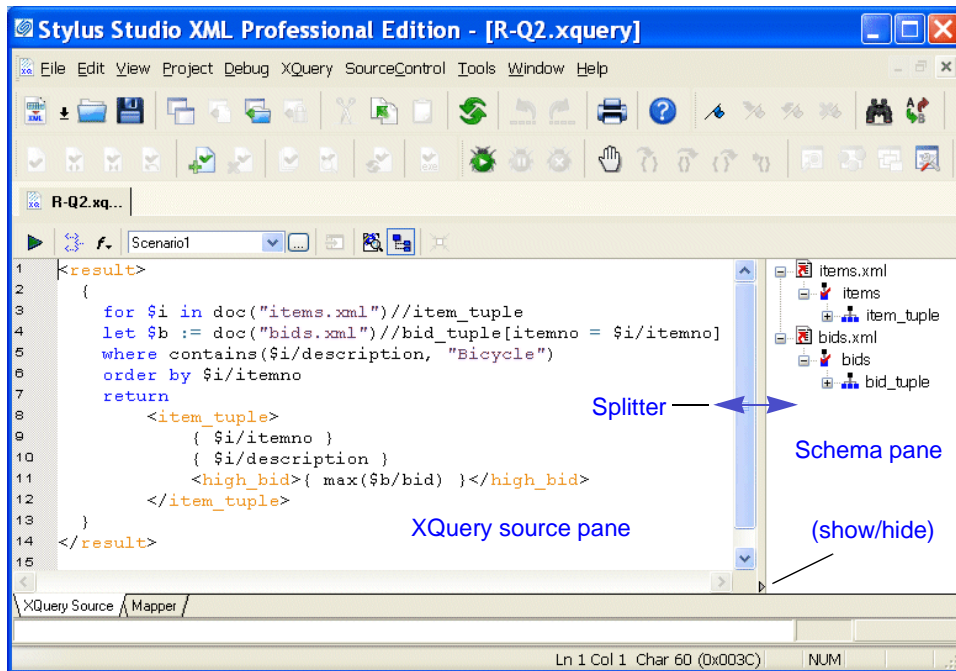
### XQuery Source Tab

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:

- A source pane, which shows the XQuery source, 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 source 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 source 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 296. 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.

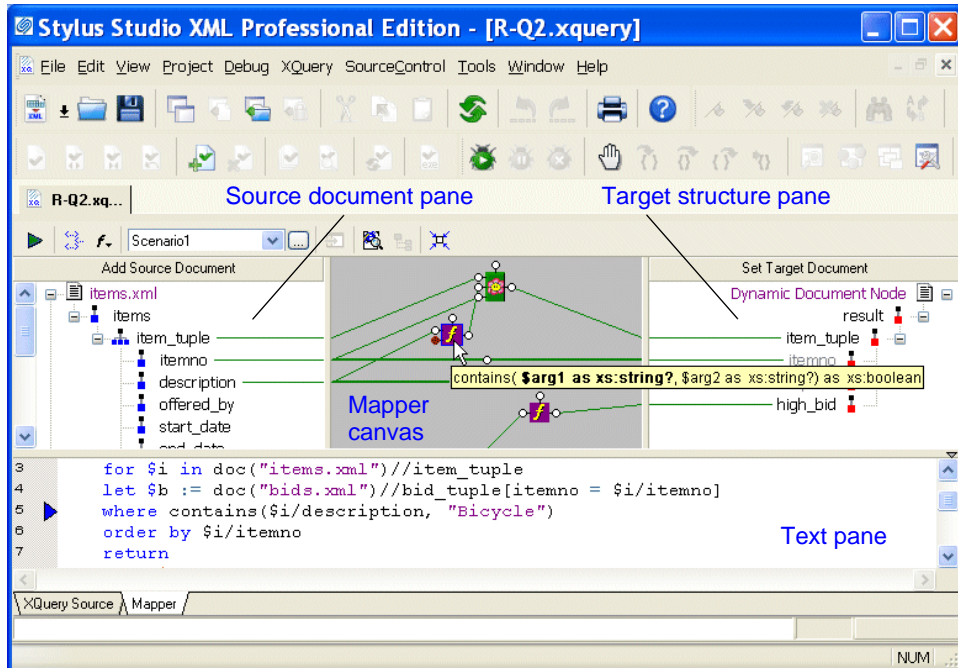
You can define other XQuery editor settings on the **Editor General** and **Editor Format** pages of the **Options** dialog box. (Click **Tools > Options**.)

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 297. 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.
- Source code pane (not shown in [Figure 297](#)). 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 **XQuery Source** tab. Of course, the **XQuery Source** tab is available if you prefer working with the source using a full-page view. All views – Mapper tab, XQuery Source tab, and the source pane – are synchronized. When displayed, the source pane spans the width of the XQuery editor.

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.

For more information

See [“Building an XQuery Using the Mapper”](#) on page 700 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.

## Building an XQuery Using the Mapper

This section describes how to build a new XQuery using Stylus Studio’s XQuery mapper. This section covers the following topics:

- [“Process Overview”](#) on page 701
- [“Source Documents”](#) on page 701
- [“Specifying a Target Structure”](#) on page 707
- [“Modifying the Target Structure”](#) on page 709
- [“Mapping Source and Target Document Nodes”](#) on page 711
- [“FLWOR Blocks”](#) on page 716
- [“Function Blocks”](#) on page 719

- [“IF Blocks”](#) on page 722
- [“Condition Blocks”](#) on page 723

## 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.

## 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.

In this section

This section covers the following topics:

- [“Choosing Source Documents”](#) on page 702
- [“Source Documents and XML Instances”](#) on page 702
- [“How to Add a Source Document”](#) on page 704
- [“How to Remove a Source Document”](#) on page 706
- [“How Source Documents are Displayed”](#) on page 706

### 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.

For more information

See [“Source Documents and XML Instances”](#) on page 702 to learn more about how Stylus Studio treats source documents. See [“Creating an XQuery Scenario”](#) on page 731 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.

Association types

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:

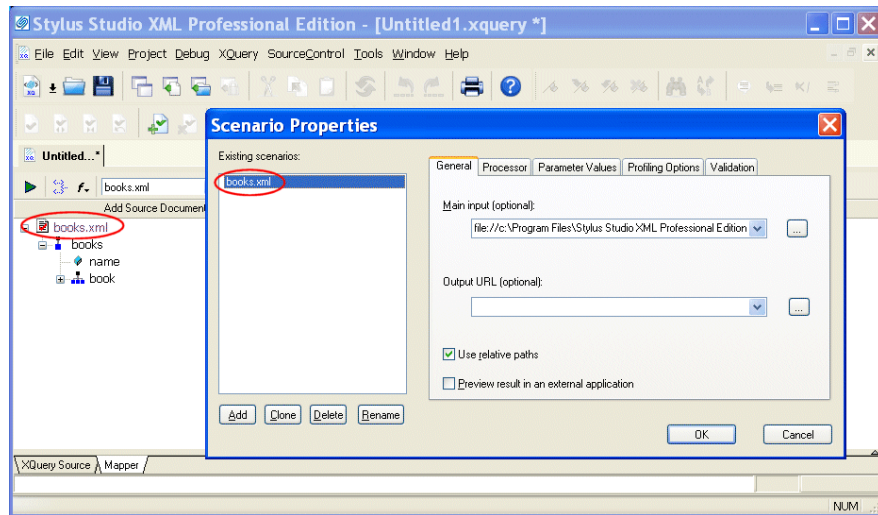
```
for $book in document("file:///c:\Program Files\Stylus Studio\examples\simpleMappings\catalog.xml")/books/book
```

- 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 298. 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 731.

- 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 75. 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 <a href="#">“How to Change a Source Document Association”</a> on page 704.
	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

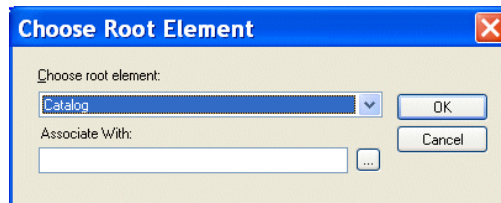
**To add an XQuery source document to XQuery mapper:**

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 299. 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 716.

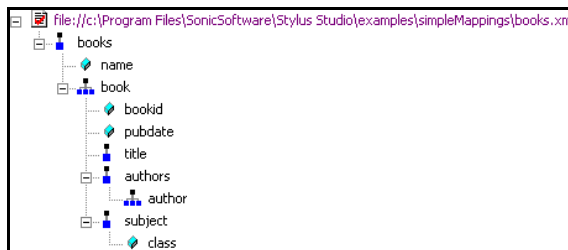
#### To remove a source document from XQuery:

1. Remove any maps from the source document to the target schema. (See [“Removing Source-Target Map”](#) on page 716 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 702 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 300. 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 76.**

<i>Symbol</i>	<i>Meaning</i>
	Repeating element
	Element
	Attribute

See “[Source document icons](#)” on page 704 to learn about the different ways source document icons are depicted.

## 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.
- 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 708
- “[Building a Target Structure](#)” on page 708

For more information

See “[Modifying the Target Structure](#)” on page 709 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.

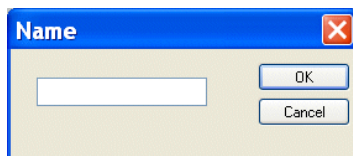


Figure 301. Name Dialog Box

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 302. 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 709
- [“Removing a Node”](#) on page 710
- [“Setting a Text Value”](#) on page 710

### Adding a Node

See [“How to create elements and attributes”](#) on page 709.

### 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 716.

#### 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 716 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.

**Example** Here is the XQuery code Stylus Studio generates for the `Title` element when a text value is specified for it:

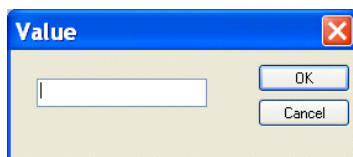
```
<Book>
  <Title>Confederacy of Dunces</Title>
</Book>
```

Stylus Studio displays a red letter **T** for nodes for which you define a text value: `Title` **T**.

How to set a text value

#### 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 303. 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 711
- [“Left and Right Mouse Buttons Explained”](#) on page 711
- [“How to Map Nodes”](#) on page 712
- [“Link Lines Explained”](#) on page 713
- [“Removing Source-Target Map”](#) on page 716

### 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 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).

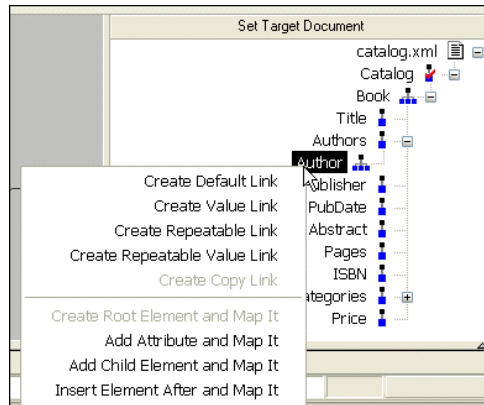
As you use the splitter in the XQuery mapper to widen the source and target document panes, the size of the mapper canvas is reduced. The **Fit in Mapper Canvas** button () , located at the top of the XQuery mapper, redraws the diagram in whatever space is currently available to the mapper canvas. This feature is also available from the mapper short-cut menu (right-click anywhere on the mapper canvas to display the short-cut menu).

### 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 304. 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`:

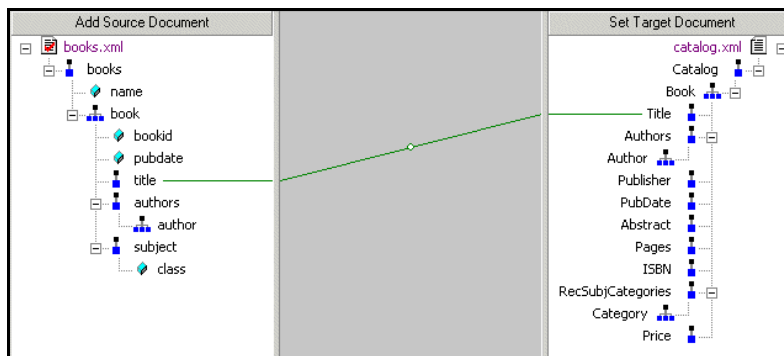


Figure 305. Thin Lines in XQuery Mapper

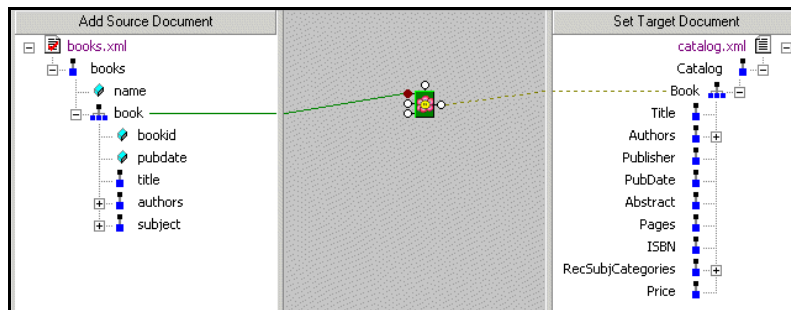
This map results in Stylus Studio composing the following XQuery code:

```
<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 306. 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:

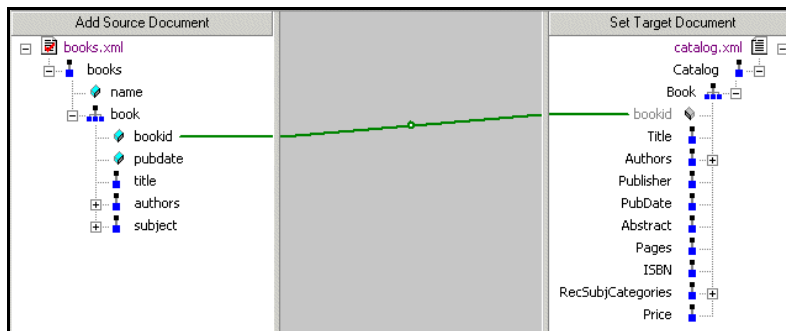


Figure 307. Thick Lines in XQuery Mapper

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).

### 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 717
- [“Creating a FLWOR Block”](#) on page 718

## 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:

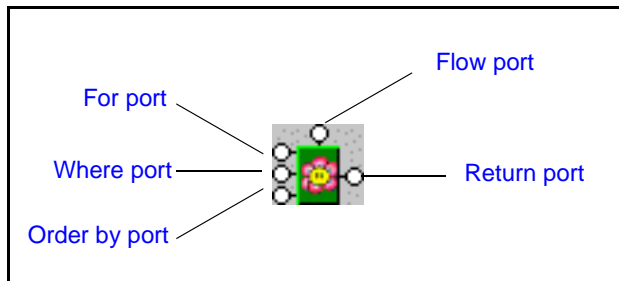


Figure 308. FLWOR Block

### 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 718). 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 722 and “[Function Blocks](#)” on page 719 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 719
- [“Parts of a Function Block”](#) on page 720
- [“User-Defined Functions”](#) on page 721
- [“concat Function Blocks”](#) on page 722

### 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 721.

### 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:

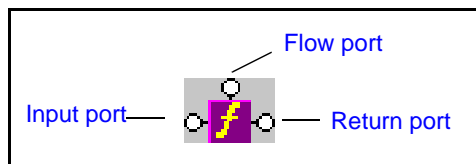
1. Right-click on the mapper canvas.
2. Select **New > User Functions** from the shortcut menu.

Any user-defined functions defined in the XQuery source are displayed in a sublist.

See “[User-Defined Functions](#)” on page 721 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 718.

### 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

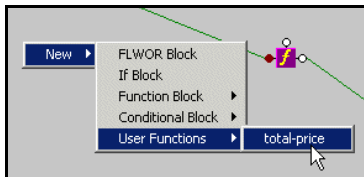
You can define your own functions in XQuery, such as the following:

```
define 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**.

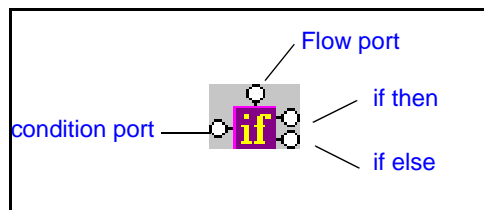


Figure 314. IF Block



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.

## Debugging XQuery Documents

Complex XQuery documents require robust debugging features.

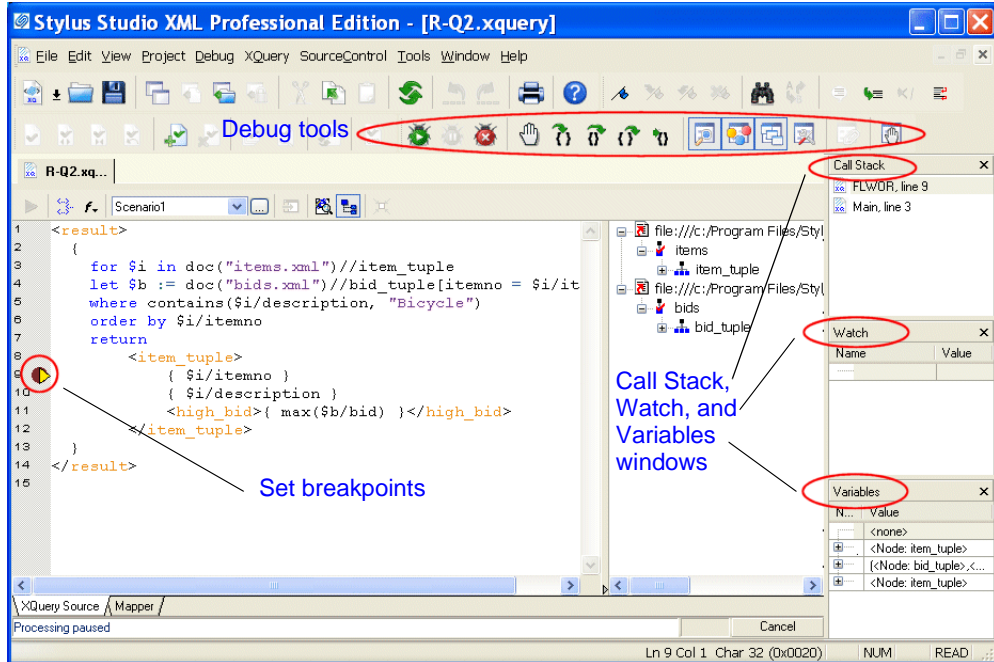


Figure 316. Debugging XQuery in Stylus Studio

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.

In this section

This section covers the following topics:



- “Using Breakpoints” on page 725
- “Viewing Processing Information” on page 726
- “Using Bookmarks” on page 728
- “Profiling XQuery Documents” on page 728

## 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 726) 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 726
- [“Evaluating XPath Expressions in the Current Processor Context”](#) on page 726
- [“Obtaining Information About Local Variables”](#) on page 727
- [“Displaying a List of Process Suspension Points”](#) on page 727
- [“Displaying XQuery Expressions for Particular Output”](#) on page 727


## 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 Edition.

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:

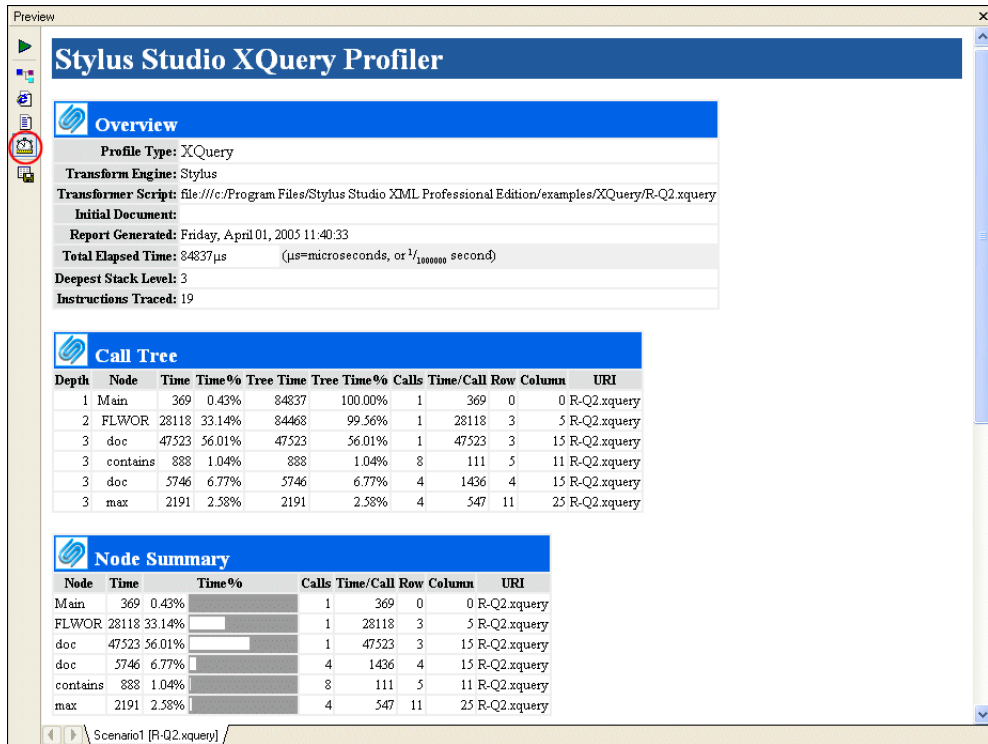


Figure 317. XQuery Profiler Report

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 730 for more information about this procedure.

### 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

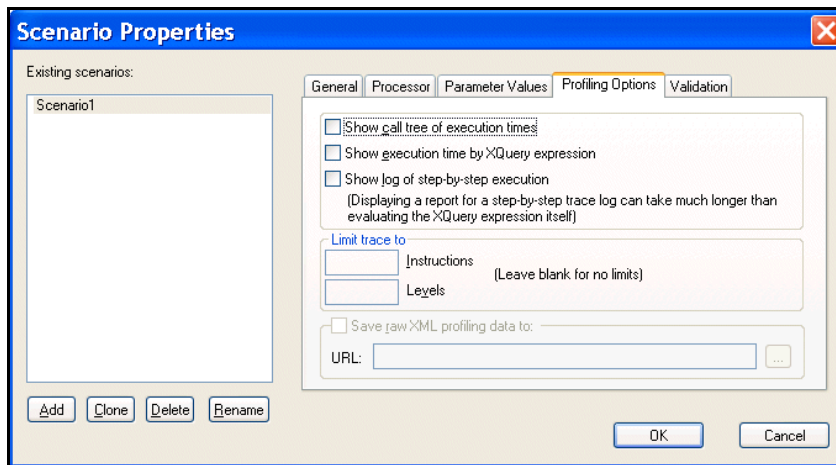
**Note** Displaying the report for a step-by-step log can take significantly longer than evaluating the XQuery itself. Consider using the Profiler with the first two performance metric options. You can also use the **Limit Trace To** fields to further restrict the Profiler's scope. If you find you need 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 318. 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 730 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.

## Creating an XQuery Scenario

An *XQuery scenario* is a group of settings that you can associate with an XQuery. Stylus Studio uses scenario settings each time you preview the XQuery result. For example, if you specify an XML input document in your scenario, Stylus Studio runs the XQuery against that document when you click the **Preview Result** button ()

You can create multiple scenarios that use the same XQuery, and choose different settings for each. This flexibility can aid the XQuery development process as it enables you to easily test different applications of the XQuery before you put it online.

This section covers the following topics:

- [“Overview of Scenario Features”](#) on page 732
- [“How to Create a Scenario”](#) on page 737
- [“How to Run a Scenario”](#) on page 738
- [“How to Clone a Scenario”](#) on page 738

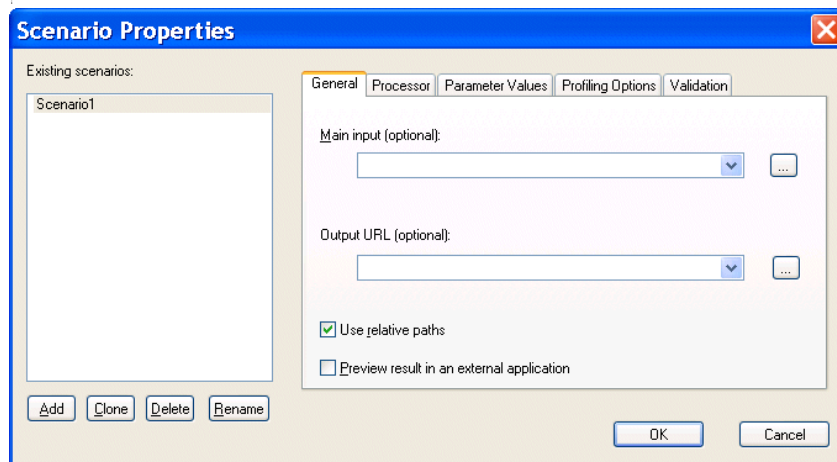
## Overview of Scenario Features

This section describes the main features of XQuery scenarios. It covers the following topics:

- “[Specifying XML Input](#)” on page 732
- “[Selecting an XQuery Processor](#)” on page 733
- “[Setting Values for External Variables](#)” on page 734
- “[Performance Metrics Reporting](#)” on page 735
- “[Validating XQuery Results](#)” on page 735

### Specifying XML Input

One benefit of the XQuery scenario feature is that it lets you test your XQuery against any number of XML documents. The XML against which you run your XQuery is referred to as *XML input*. If you build your XQuery using the XQuery Mapper, Stylus Studio uses the first source document you select as the main input XML document. You can specify XML input on the **General** tab of the **Scenario Properties** dialog box.

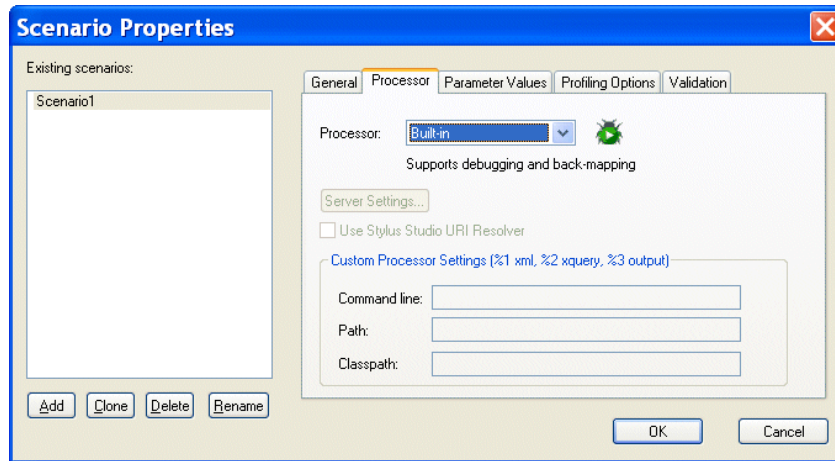


**Figure 319. XQuery Scenario General Properties**

See “[Source Documents](#)” on page 701 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 320. XQuery Scenario Processor Properties**

You can use

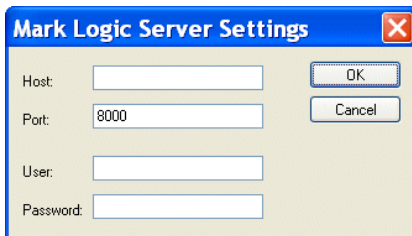
- One of several processors that support XQuery debugging and backmapping, including Stylus Studio's built-in XQuery processor
- An external instance of the Stylus Studio processor
- Mark Logic's XQuery processor, for use with documents stored in the Mark Logic Content Interaction Server
- A custom processor

### Using the Mark Logic Processor

If you want to use the Mark Logic processor:

1. Click **Mark Logic**.  
The **Server Settings** button becomes active.
2. Click the **Server Settings** button.

The **Mark Logic Server Settings** dialog box appears.



**Figure 321. Mark Logic Server Settings Dialog Box**

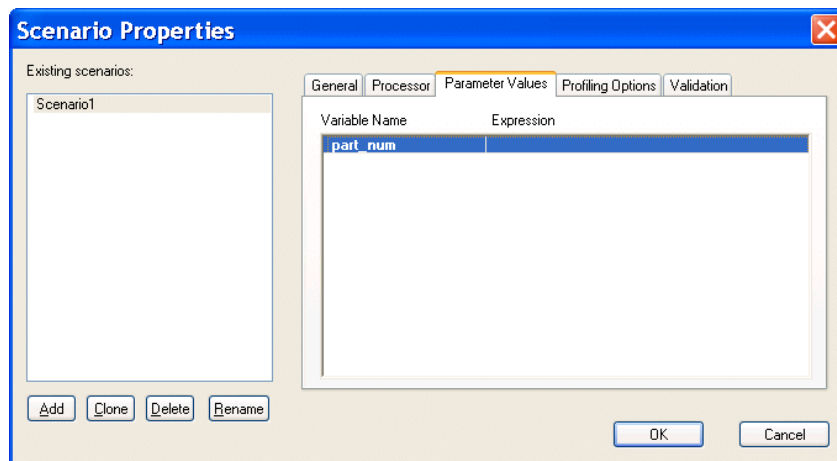
3. Enter the host, port, username, and password information for the server on which the Mark Logic Content Interaction Server is running.
4. Click **OK**.

### 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 322. 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 730 to learn more about the different ways in which Stylus Studio can provide you with XQuery performance metrics.

### Validating XQuery Results

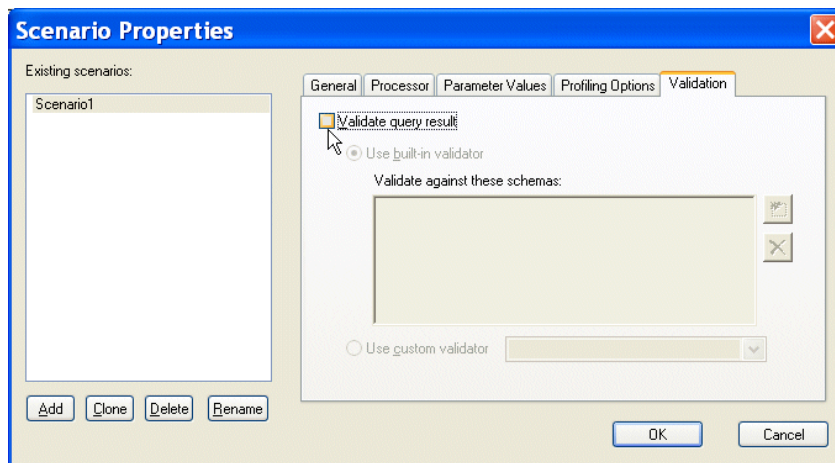
You can optionally validate the XML document that results from XQuery processing. You can validate using the

- 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, Xerces-J, 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 323. Validation Tab for XQuery 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**.

## 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. 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.
5. If you want Stylus Studio to store paths relative to XQuery document path, ensure that the **Use relative paths** option is checked.
6. 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.
7. 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.
8. If you want Stylus Studio to capture performance metrics, enable the XQuery Profiler on the **Profiling Options** tab. See [Profiling XQuery Documents](#) on page 728.
9. 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 738.
10. 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 737.
  7. Click **OK**.
- If you change your mind and do not want to create the clone, click **Delete** and then **OK**.



## Generating Java Code for XQuery

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.

This section covers the following topics:

- [“Scenario Settings”](#) on page 739
- [“Java Code Generation Settings”](#) on page 741
- [“How to Generate Java Code for XQuery”](#) on page 742
- [“Compiling Generated Code”](#) on page 743

**Tip** You can also generate Java code for XSLT. See [“Generating Java Code for XSLT”](#) on page 399.

### 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 77. Scenario Settings that Affect Java Code Generation**

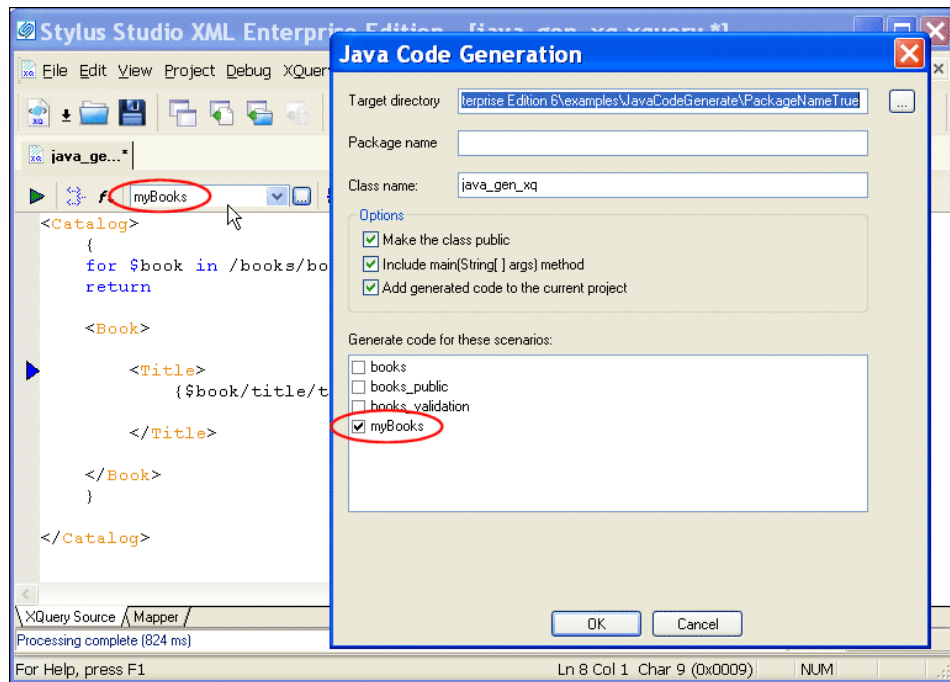
<i>Tab</i>	<i>Comment</i>
General	The Java Code Generation wizard uses only the <b>Main input</b> and <b>Output URL</b> fields, if specified.
Processor	You must use the Saxon processor. The Stylus Studio URI Resolver is also used to resolve non-standard URIs if you select the check box on the <b>Processor</b> page.
Parameter Values	Any parameter values you specify are processed by the Java Code Generation wizard.

**Table 77. Scenario Settings that Affect Java Code Generation**

Tab	Comment
Profiling Options	Ignored.
Validation	Validation is optional. If you choose to validate XQuery output, the Java Code Generation wizard always uses the Xerces-J processor, regardless of the validator you specify on the <b>Validation</b> tab. If you want to specify external schemas for validation purposes, click <b>Use built-in validator</b> . Note that the Xerces-J 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 324](#).



**Figure 324. 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

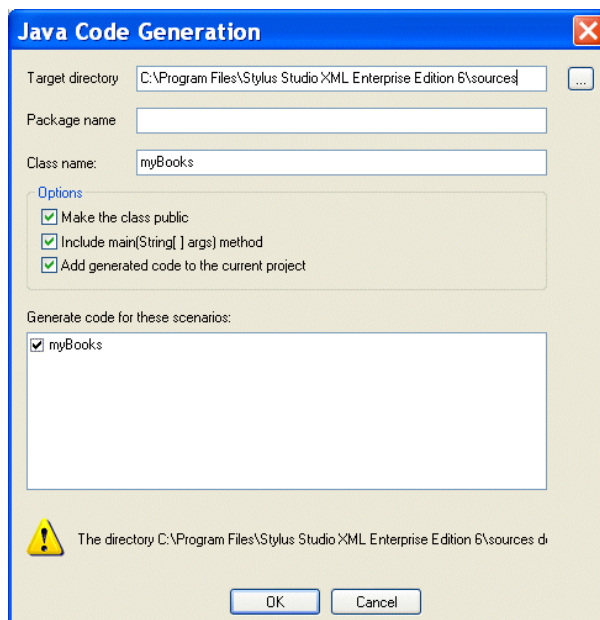
All of these options are selected by default.

**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 processor. See “[Scenario Settings](#)” on page 739 for more information.
2. Select **XQuery > Generate Java Code** from the Stylus Studio menu. The **Generate Java Code** dialog box appears.



**Figure 325. 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 741 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 (*cclassname.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:

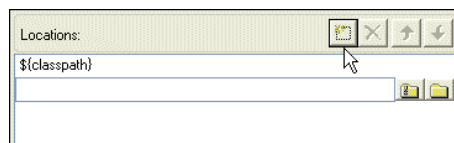
- CustomFileSystem.jar
- Saxon8.jar

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:

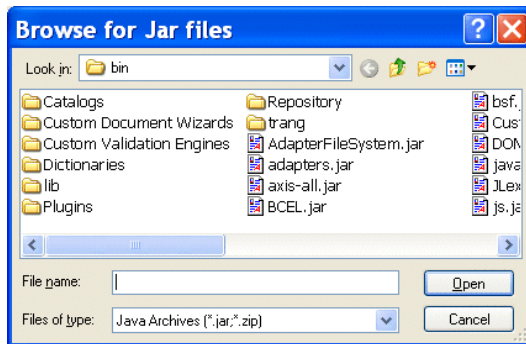
1. Select **Tool > Options** from the Stylus Studio menu.
2. Click **Java Virtual Machine** under **General**.
3. Click the browse button alongside the **Classpath** field.  
The **Add Directory or JAR File to Classpath** dialog box appears.
4. Click the browse folders (  ) button.  
A new entry field appears in the **Locations** list box. Two buttons appear to the right of the entry field.



**Figure 326. Entry Field for JAR File Classpath**

5. Click the browse jar files button (  ).

Stylus Studio displays the **Browse for JAR Files** dialog box.




**Figure 327. 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.

## Chapter 11    **Composing Web Service Calls**



The Web services features described in this chapter are available only in Stylus Studio XML Enterprise Edition.

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.

For more information

Stylus Studio provides a video demonstration for Stylus Studio's Web services features. Visit our web site to view this and other Stylus Studio video demonstrations:

[http://www.StylusStudio.com/xml\\_videos.html](http://www.StylusStudio.com/xml_videos.html)

Contents

This chapter covers the following topics:

- “Overview” on page 746
- “Obtaining WSDL URLs” on page 748
- “Modifying a SOAP Request” on page 752
- “Testing a Web Service” on page 754
- “Saving a Web Service Call” on page 755
- “Creating a Web Service Call Scenario” on page 758

### 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 748.
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 752.
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 754.
4. Optionally, save the Web service call for later use. See [“Saving a Web Service Call”](#) on page 755.
5. Optionally, create a Web service scenario. See [“Creating a Web Service Call Scenario”](#) on page 758.

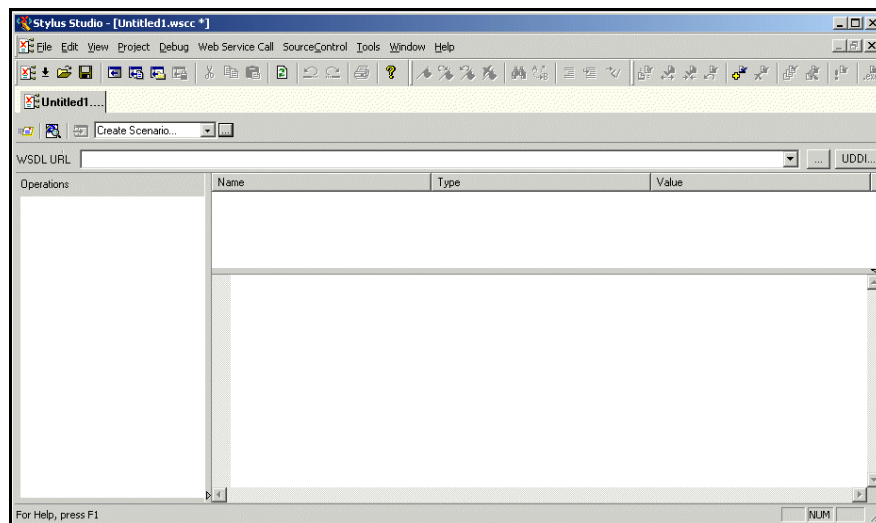
### How to Compose a Web Service Call

◆ **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 328. 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 748 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.

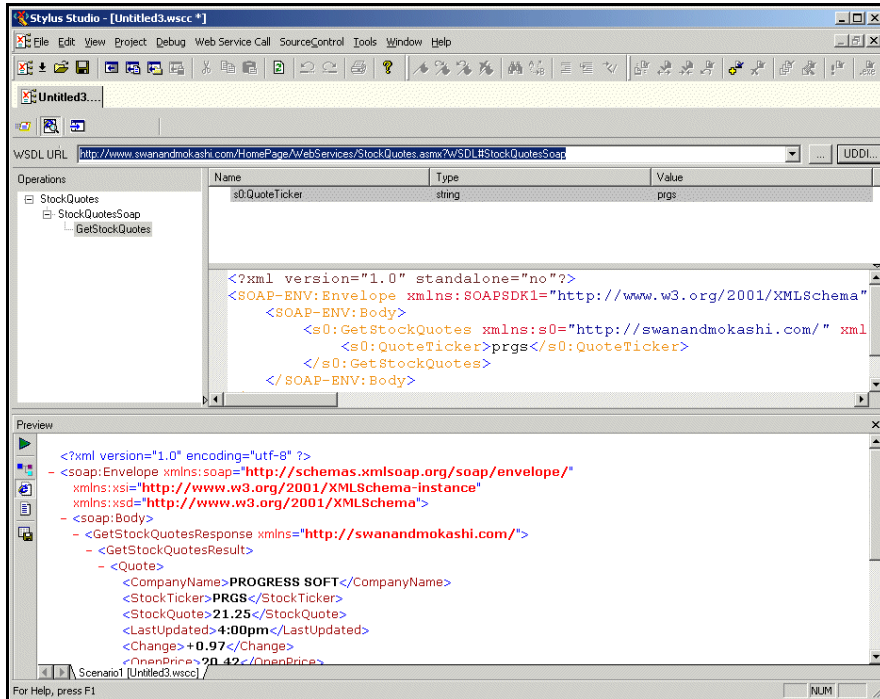
Stylus Studio updates the SOAP request to reflect the parameter values you enter.

*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 752 for help with this step.

- 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 329](#):



**Figure 329. SOAP Response**

- Optionally, save the Web service call. See [“Saving a Web Service Call”](#) on page 755 for help with this step.

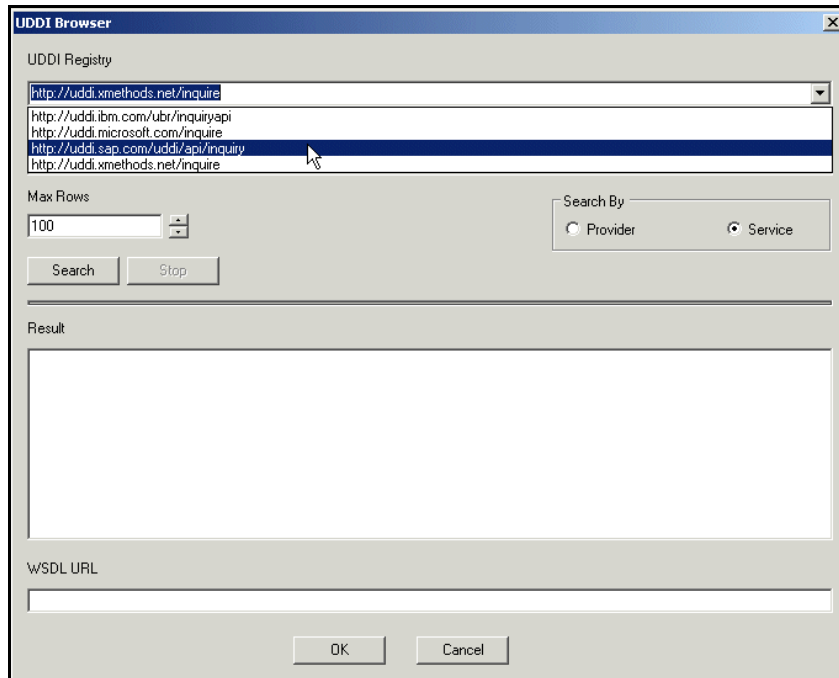
## Obtaining WSDL URLs

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 330. 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.

## Composing Web Service Calls

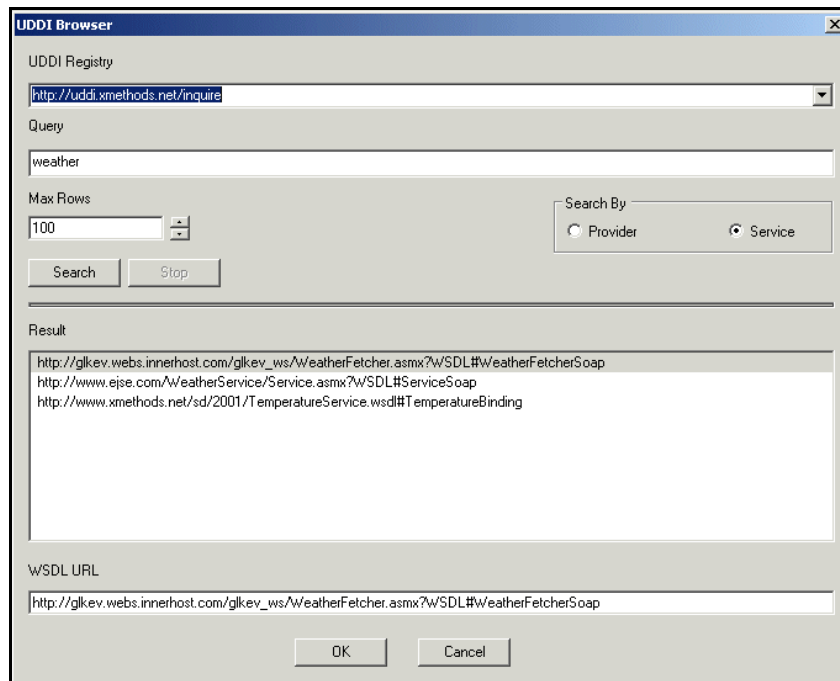
---

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 331. UDDI Browser result**

What to do  
next

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 752.

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.

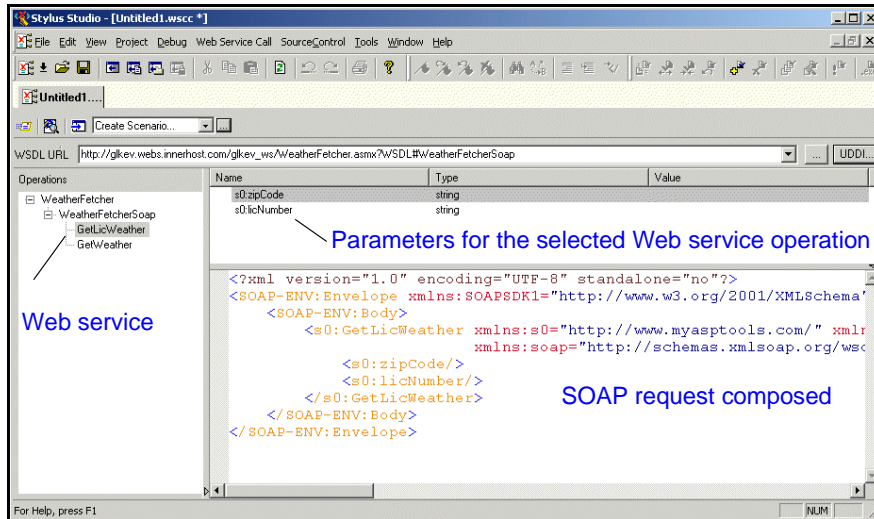
## How to Browse UDDI Registries

### ◆ 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 332. 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 333](#):

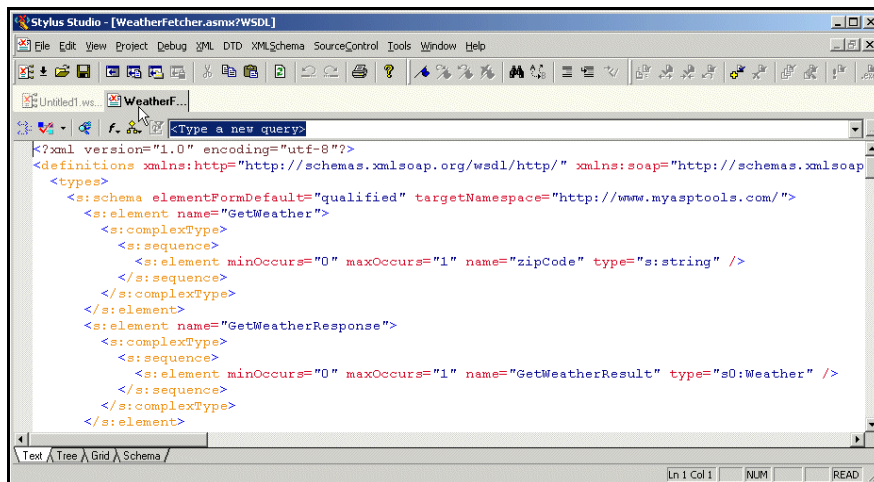


Figure 333. WSDL document editor

## 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.

What to do next      Once you have specified values for the SOAP request's parameters, you can test the Web service. See [“Testing a Web Service”](#) on page 754.

## 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

- 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 758 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 334:

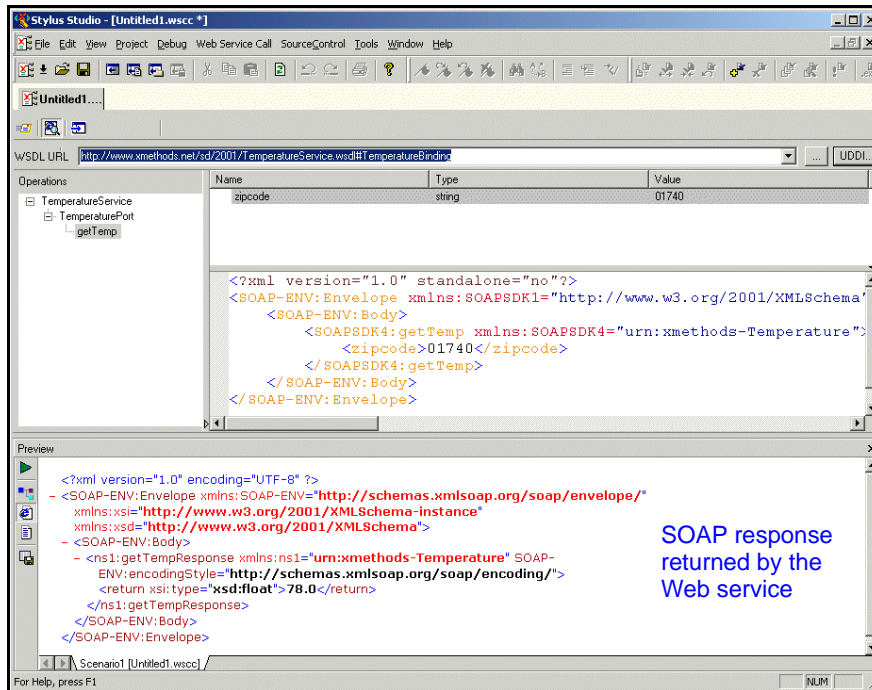


Figure 334. Testing a Web service

## 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:

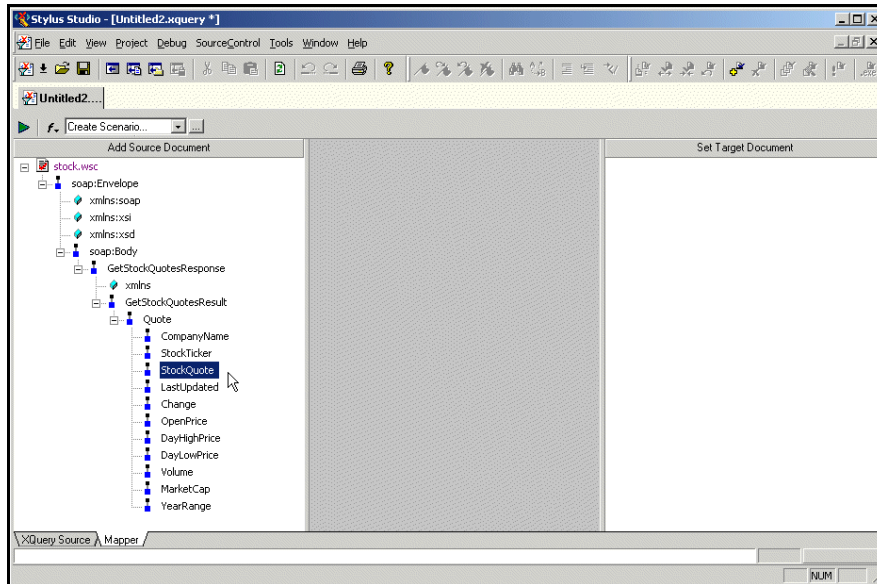
```
<?xml version="1.0" standalone="no"?>
<SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  SOAP-ENV:Body>
    <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 335](#):

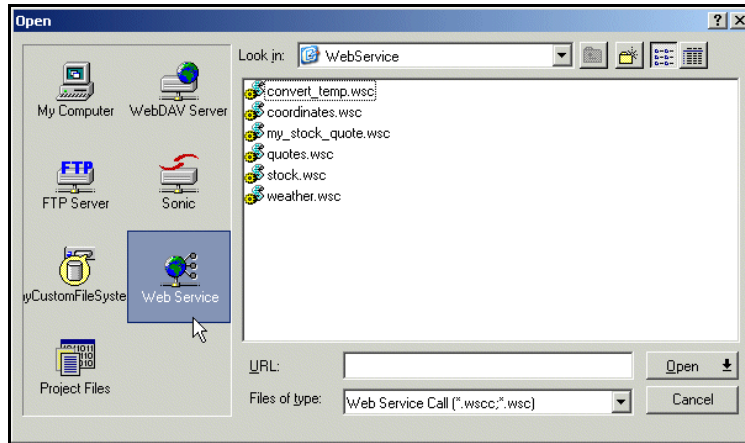


**Figure 335. Using a Web Service Call to Compose an XQuery**

XQueries composed using a Web service call as a source document return real-time data from the Web service as a result.

### Where Web Service Calls are Saved

Stylus Studio saves Web service calls to the Web Services file system. By default, Web service calls are saved with a `.wsc` extension. When you open a saved Web service call, you must look in the Web Services file system, as shown in [Figure 336](#):



**Figure 336. Open Dialog Box**

### 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**.

### 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 call 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.

When to  
create a  
scenario

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 759
- [“How to Create a Scenario”](#) on page 762
- [“How to Run a Scenario”](#) on page 763
- [“How to Clone a Scenario”](#) on page 764

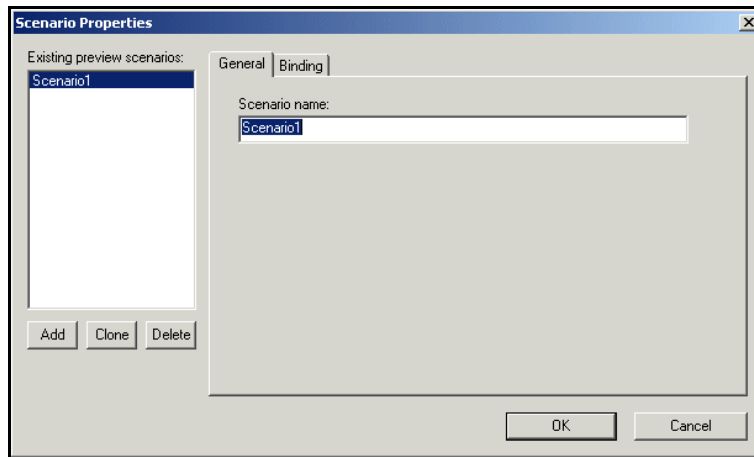
### Overview of Scenario Features

This section describes the main features of Web service call scenarios. It covers the following topics:

- [“Scenario Names”](#) on page 760
- [“Transport Protocol and Client Settings”](#) on page 761
- [“Other Transport Settings”](#) on page 761

### Scenario Names

You specify a name for a Web service call scenario on the **General** tab of the **Scenario Properties** dialog box.

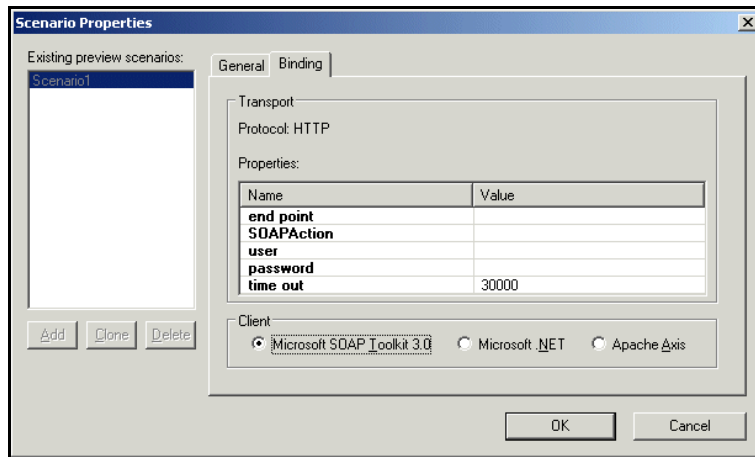


**Figure 337. General Tab of the Web Service Call Scenario Properties Dialog Box**

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.



**Figure 338. Binding Tab of the Web Service Call 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 SOAP Toolkit
- 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 78. HTTP Settings**

<i>Setting</i>	<i>Description</i>
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.

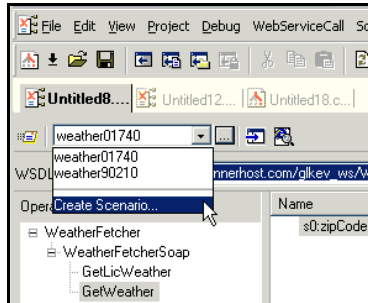
## How to Create a Scenario

◆ **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 746 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 339. Creating a Scenario**

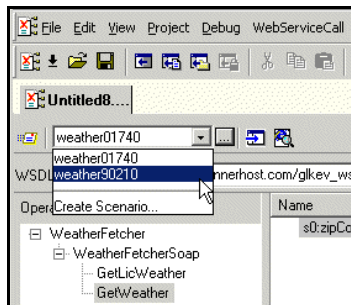
3. On the **General** tab, specify a name for the Web service call scenario.
4. Click the **Binding** tab.
5. Select the appropriate transport protocol from the **Transport** drop-down list.
6. Specify the binding properties you want to associate with this Web service call scenario.
7. Click **OK**.

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 340. 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 759.
6. Click **OK**.  
If you change your mind and do not want to create the clone, click **Delete** and then **OK**.

## Chapter 12    **Accessing Relational Data as XML**



The features for working with relational data as XML described in this chapter are available only in Stylus Studio XML Enterprise Edition and Stylus Studio XML Professional Edition.

This chapter describes the different ways you can access relational data and render that data as XML in Stylus Studio:

- DB-to-XML data sources, which you can use to query and, optionally, to modify data in a relational database from within Stylus Studio
- The URL Builder, which lets you create an XML document based on a table or view you select from a relational database

Stylus Studio provides a video demonstration of Stylus Studio's DB-to-XML data source feature. Visit our Web site to view this and other Stylus Studio video demonstrations:

[http://www.StylusStudio.com/xml\\_videos.html](http://www.StylusStudio.com/xml_videos.html)

This chapter covers the following topics:

- “[Overview of DB-to-XML Data Sources](#)” on page 766
- “[Creating a DB-to-XML Data Source](#)” on page 770
- “[Specifying Connection Settings](#)” on page 773
- “[Working with Scenarios](#)” on page 776
- “[Composing SQL/XML in Stylus Studio](#)” on page 780
- “[Working with Relational Data as XML](#)” on page 784
- “[Using the URL Builder](#)” on page 789

# Overview of DB-to-XML Data Sources

SQL/XML is an extension of the Structured Query Language (SQL) that allows you to query relational data, render query results as XML, and write modified or new data back to a relational database. Stylus Studio supports the SQL/XML standard by allowing you to create a *DB-to-XML data source*, a file that contains

- Connection settings used to connect to a specific relational database on a specific server. You specify connection settings in a scenario. See [“DB-to-XML Data Source Scenarios”](#) on page 767 for more information.
- SQL/XML statements (SELECT and UPDATE, for example) that you want to execute on that database. Query results returned by the DB-to-XML data source you define can be used in Stylus Studio anywhere you use XML.

If you need only to render an entire table or view as XML, use the URL Builder. Any time you need more control – writing your own SQL/XML or updating the relational database, for example – use a DB-to-XML data source. See [“Using the URL Builder”](#) on page 789 for more information on the URL Builder Utility.

This section covers the following topics:

- [“System Requirements”](#) on page 766
- [“Supported Databases”](#) on page 766
- [“DB-to-XML Data Source Scenarios”](#) on page 767
- [“The DB-to-XML Data Source Editor”](#) on page 768

## System Requirements

In order to use DB-to-XML data sources, the Java Runtime Environment (JRE) 1.4.x or greater must be installed on your machine. This feature also requires that you install the JRE with the optional module, Support for Additional Languages. You can verify whether or not this module has been installed by checking your JRE for these files:

- `i18n.jar`
- `charsets.jar`

## Supported Databases

Stylus Studio supports several relational databases, including:

- IBM DB2
- Informix

- Microsoft SQL Server
- Oracle
- Sybase

In addition, you can use DataDirect SequeLink Server, bundled with Stylus Studio, to connect to any ODBC data source.

See [“Choosing a Database”](#) on page 773 to learn more about specifying a database for a DB-to-XML data source.

### DB-to-XML Data Source Scenarios

A *DB-to-XML data source scenario* is a group of customizable settings associated with a DB-to-XML data source. You use a scenario to specify connection settings, such as the

- Database type
- Database server URI
- Username and password required to log on to the database server

You define these settings on the **Scenario Properties** dialog box. See [“Specifying Connection Settings”](#) on page 773 to learn more about DB-to-XML data source scenarios.

## The DB-to-XML Data Source Editor

You use the DB-to-XML Data Source Editor to specify database connectivity settings, and to compose the SQL/XML (queries and other actions) that you want to execute on the server.

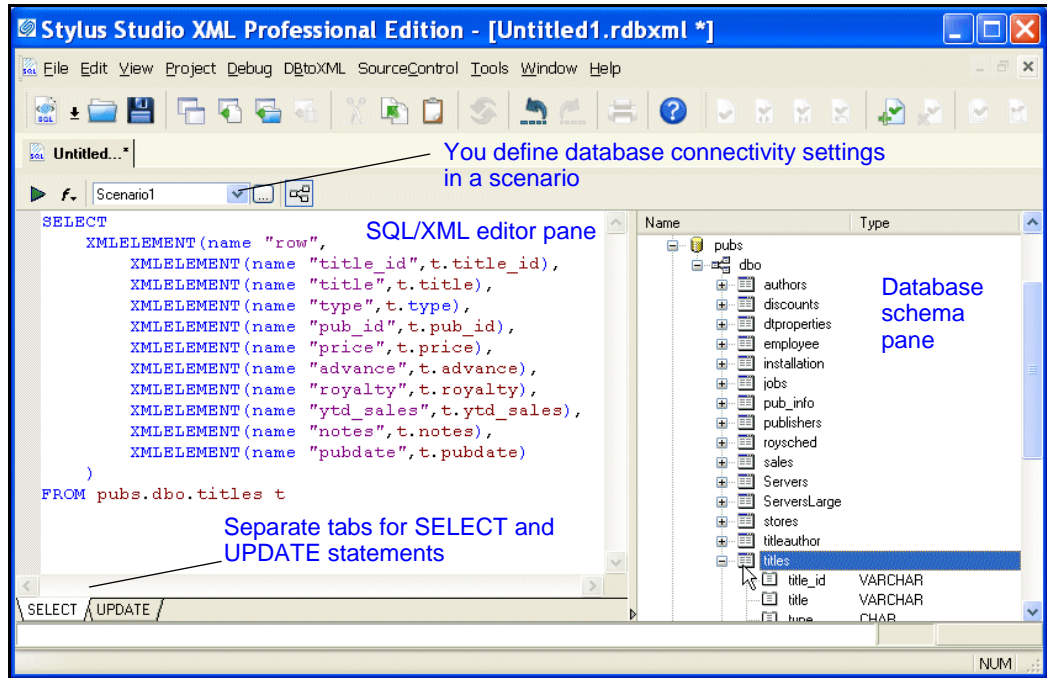


Figure 341. DB-to-XML Data Source Editor

**Tip** Once you connect to a server, Stylus Studio displays the databases, tables, views, and columns on the right side of the editor. When composing your SQL statements, you can drag and drop tables, views, and columns from the database schema tree into the SQL/XML editor.

### SQL/XML Editor Pane

The SQL/XML editor pane is on the left side of the DB-to-XML Data Source Editor. It consists of two tabs, **SELECT** and **UPDATE**, which you use to compose SQL/XML SELECT and, optionally, UPDATE and INSERT statements. You can type directly in the editor, or you can use drag-and-drop to automatically compose SQL/XML statements.

See “Composing SQL/XML in Stylus Studio” on page 780 to learn more about using the SQL/XML editor.

## Database Schema Pane

The database schema pane is on the right side of the DB-to-XML Data Source Editor. Once you have established a connection to the database specified in the scenario, Stylus Studio displays a tree diagram of the database schema pane.

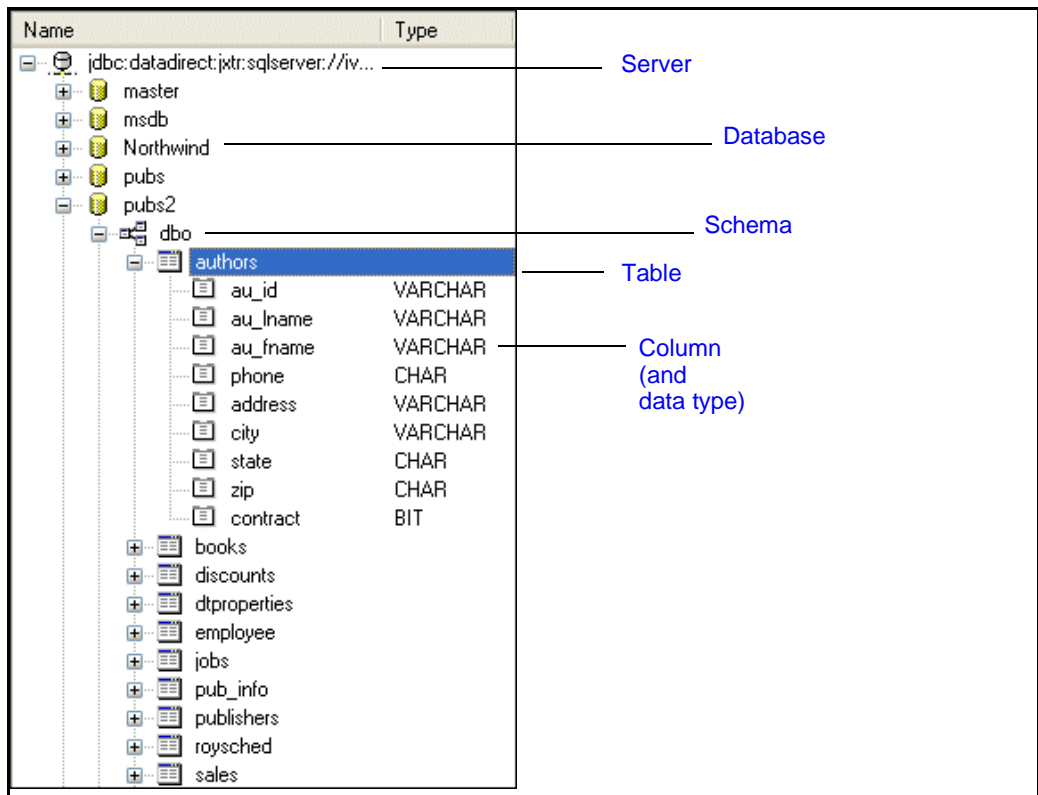



Figure 342. Database Tree Displayed in Database Schema Pane

The database schema tree is displayed by default. You can hide it by clicking the **Show/Hide Database Schema** button () , or by selecting **DB-to-XML > Database Schema** from the menu.

**Note** Once the connection with the server has been established and the database is located, Stylus Studio drops the connection to the server. You can reconnect to the database to refresh the database schema tree by selecting **DB-to-XML > Load Database Schema** from the menu bar.

Information in the schema tree is displayed in a hierarchical fashion, starting with the server you specified in the connection settings. Tables, views, and columns are displayed in database order; column data types are displayed in the **Type** column.

### Working with Relational Data

All SQL/XML statements are defined on the **SELECT** and **UPDATE** tabs. Stylus Studio can automatically create valid SELECT, INSERT, and UPDATE SQL/XML statements based on the objects you select from the database schema tree, or you can type your own SQL/XML. The actions that can be performed by the SQL/XML you define in your DB-to-XML data source are limited only by the permissions associated with the username you use to log in to the database. See “[Composing SQL/XML in Stylus Studio](#)” on page 780 for more information on this topic.

## Creating a DB-to-XML Data Source


### ◆ To create a DB-to-XML data source:

1. Select **File > New > DB to XML Data Source** from the menu. Stylus Studio displays the **Scenario Properties** dialog box.
2. Specify a scenario name (or accept the default).
3. Specify the information needed to connect to the database server and log in to the database. This information includes
  - The database type (Oracle or Sybase, for example)
  - The server on which the database resides
  - The username, password, and any other information required to log in to the database.

See “[Specifying Connection Settings](#)” on page 773 if you need help with this step.



*Alternative:* If you have already defined a default connection setting and you want to use it for this scenario, click the **Use Default Connection** field.

4. Click **OK** to save the scenario.  
Stylus Studio automatically connects to the database and logs you into the server. The database schema, table, view, and column information for the database you specified in the scenario appears in the schema pane of the DB-to-XML Data Source Editor.
5. Compose the SQL/XML SELECT statement to query the database. (UPDATE and INSERT statements, which write modified data back to the database, are optional.) See “[Composing SQL/XML in Stylus Studio](#)” on page 780.
6. Click the **Execute Query** button (  ) the query and review the XML it returns.
7. Optionally (if you have written SQL/XML to write new or modified data to the database), save the XML document returned by the SQL/XML query to write data back to the database. See “[Working with Relational Data as XML](#)” on page 784.

The remainder of this chapter describes these steps in greater detail.

## Saving a DB-to-XML Data Source

DB-to-XML data sources are saved with a `.rdbxml` extension to their own file system, **Relational DB**.

- ◆ **To save a DB-to-XML data source:**
  1. Click **File > Save** or **File > Save As** from the menu.  
The **Save As** dialog box appears. The **Look in** field defaults to the Relational DB file system.
  2. Specify a name for your data source.
  3. Click **Save**.

## Opening a DB-to-XML Data Source

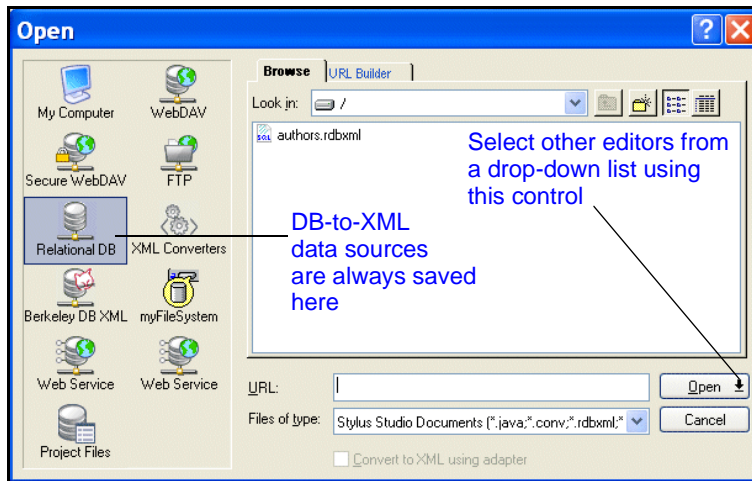
You can open a DB-to-XML data source

- As a DB-to-XML data source, that is, in the DB-to-XML data source editor. This gives you the opportunity to change the data source’s definition, including its connectivity settings and SQL/XML statements, for example.

- As XML, using any Stylus Studio editor that accepts XML (such as the XML editor, for example). When you open a DB-to-XML data source as XML, Stylus Studio displays only the XML that resulted from executing the SQL/XML defined on the **SELECT** tab of the DB-to-XML data source.

◆ **To open a DB-to-XML data source:**

1. Select **File > Open** from the menu bar.  
Stylus Studio displays the **Open** dialog box.



**Figure 343. Open Dialog Box**

2. Click the **Relational DB** icon to display the files in the DB-to-XML file system.
3. Select the data source you want to open.
4. If you want to open the data source in the DB-to-XML Data Source Editor, click **Open**.

If you want to open the data source as an XML document:

- a. Click the down arrow on the **Open** button.
- b. Select the editor from the drop-down list.

Stylus Studio opens the data source as an XML document in the editor you selected.

**Note** When you save a DB-to-XML data source that you have opened as XML, any SQL/XML statements you have defined on the **UPDATE** tab are executed.

## Specifying Connection Settings

Stylus Studio uses connection settings to connect to the database from which you want to create your DB-to-XML data source. Specific properties vary from database to database, but you generally 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), or you can use DataDirect SequeLink to connect to any ODBC database you specify.
- 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 tells you how to create a default connection setting you can reuse with any DB-to-XML data source you create.

In this section

This section covers the following topics:

- [“Choosing a Database”](#) on page 773
- [“Using the Server URL Field”](#) on page 774
- [“Username and Password”](#) on page 775
- [“Creating a Default Database Connection”](#) on page 775

### 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.
- Any ODBC-compliant database configured as a DataDirect SequeLink server.

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. If you want to use a database that is not natively supported, select **DataDirect SequeLink**.

### Using SequeLink

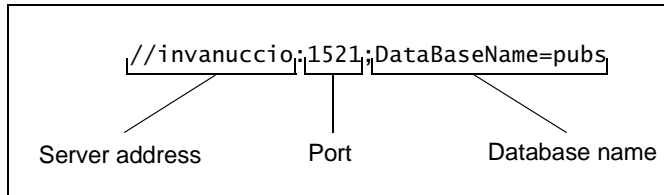
You can use DataDirect SequeLink to connect to any database that complies with the Microsoft Open Database Connectivity (ODBC) specification. This process involves:

1. Installing DataDirect SequeLink Server. (Stylus Studio is automatically configured as a SequeLink client when you install it.)
2. Selecting **DataDirect SequeLink** as the Database Type for your DB-to-XML data source.

SequeLink product software and documentation are in the \SequeLink directory where you installed Stylus Studio (c:\Program Files\Stylus Studio\SequeLink, for example). See the Stylus Studio *Release Notes* for information on installing SequeLink Server.

### 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 DB-to-XML data source.

**Note** The permissions of the user you specify dictate the types of operations that can be performed by the SQL/XML on the database. For example, if you specify a user with read-only permissions, the DB-to-XML data source will not be able to perform any INSERT or UPDATE operations.

## Creating a Default Database Connection

To connect to a database when defining a relational data source, you can

- Specify connection settings individually, for each data source you create
- Create a default connection setting, which you can then reuse for any data source you create

Default database connections use the same settings as other database connections you create in Stylus Studio.

### ◆ To create a default database connection:

1. Select **Tools > Options** from the Stylus Studio menu.  
The **Options** dialog box appears.
2. Select **Module Settings > DBtoXML > Default Connection**.  
The **Default Connection** page appears.
3. Select the database type, server URL, and username and password for the database to which you want to connect.
4. Click the **Connect** button.  
The list box on the **Default Connection** page displays the databases associated with the database server to which you connect.
5. Select the database to you want this connection to access.
6. Click **OK**.

The default connection is now available to use when specifying relational data sources using DB-to-XML or the URL Builder.

# Working with Scenarios

As described earlier in this chapter, you use a DB-to-XML data source scenario to specify database connectivity settings. You create and modify DB-to-XML data source scenarios using the **Scenario Properties** dialog box.


This section describes how to open the **Scenario Properties** dialog box (Stylus Studio opens it for you automatically when you create a new DB-to-XML data source), and how to create new and modify existing scenarios.

This section covers the following topics:

- [“Opening the Scenario Properties Dialog Box”](#) on page 776
- [“Creating a Scenario”](#) on page 777
- [“How to Modify a Scenario”](#) on page 779
- [“How to Clone a Scenario”](#) on page 779
- [“How to Rename a Scenario”](#) on page 780
- [“How to Delete a Scenario”](#) on page 780

## Opening the Scenario Properties Dialog Box

There are two ways to open the **Scenario Properties** dialog box for DB-to-XML data sources. Both allow you to work on existing DB-to-XML data source scenarios or to create new ones.

- ◆ **To open the Scenario Properties dialog box:**
  - Click  in the DB-to-XML data source editor tool bar.

- Select **Create Scenario** or an existing scenario from the scenario drop-down list at the top of the editor window.

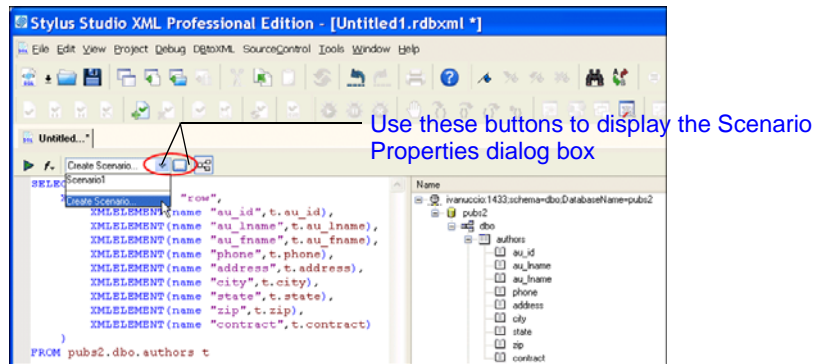


Figure 344. Ways to Open the Scenario Properties Dialog Box

## Creating a Scenario

This section describes how to create a DB-to-XML scenario.

### Naming Scenarios

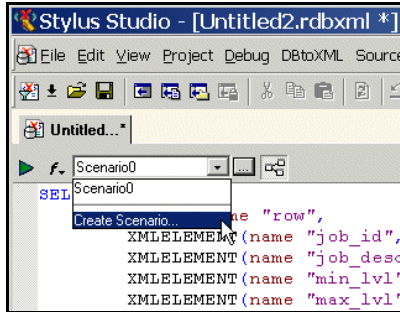
Stylus Studio supplies a default name, like Scenario0, for example, for new DB-to-XML scenarios. You can accept the default name, but you should consider typing a name that makes it easy to distinguish one scenario from another. See [“How to Rename a Scenario”](#) on page 780.

### How to Create a Scenario

#### ◆ To create a scenario:

1. Open the DB-to-XML data source editor if it is not already open.

2. Select **Create Scenario** from the scenario drop-down list at the top of the editor window:



**Figure 345. Creating a Scenario**

Stylus Studio displays the **Scenario Properties** dialog box. The name field displays a default scenario name (Scenario0, for example) in edit mode.

3. Change the default scenario name.
4. Specify the connection settings. See [Specifying Connection Settings](#) on page 773 if you need help with this step.
5. Click **OK** to save the scenario with the new settings.  
Stylus Studio connects to the server specified in the scenario.

### Alternative

Use this procedure if the **Scenario Properties** dialog box is already open.

◆ **To create a scenario when the Scenario Properties dialog box is already open:**


1. Click **New** on the **Scenario Properties** dialog box.  
The name field displays a default scenario name (Scenario0, for example) in edit mode.
2. Change the default scenario name.
3. Specify the connection settings. See [Specifying Connection Settings](#) on page 773 if you need help with this step.
4. Click **OK** to save the scenario with the new settings.  
Stylus Studio connects to the server specified in the scenario.



## How to Modify a Scenario

You can modify a scenario at any time. Change that you make take effect as soon as you save the scenario.


◆ **To modify a scenario:**

1. Display the **Scenario Properties** dialog box by clicking  in the DB-to-XML data source editor tool bar.
2. In the **Existing scenarios** field, click the name of the scenario you want to modify.
3. Change the scenario's properties as needed. See [Specifying Connection Settings](#) on page 773 if you need help with this step.
4. Click **OK** to save the new scenario.  
Stylus Studio connects to the server specified in the scenario.

## 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 DB-to-XML data source editor tool bar.
2. In the **Existing scenarios** field, click the name of the scenario you want to clone.
3. Click **Clone**.  
The name field displays a default scenario name (Scenario0, for example) in edit mode.
4. Change the default scenario name.
5. Change the source scenario's connection settings as needed. See [Specifying Connection Settings](#) on page 773 if you need help with this step.
6. Click **OK** to save the new scenario.  
Stylus Studio connects to the server specified in the scenario.


### How to Rename a Scenario

◆ **To rename a scenario:**

1. Display the **Scenario Properties** dialog box by clicking  in the DB-to-XML data source editor tool bar.
2. In the **Existing scenarios** field, click the name of the scenario you want to rename.
3. Click **Rename**.  
The name field displays the scenario name in edit mode.
4. Change the scenario's name.
5. Click **OK** to save the new scenario.  
Stylus Studio connects to the server specified in the scenario.

### How to Delete a Scenario

◆ **To delete a scenario:**

1. Display the **Scenario Properties** dialog box by clicking  in the DB-to-XML data source editor tool bar.
2. In the **Existing scenarios** field, click the name of the scenario you want to delete.
3. Click **Delete**.  
The scenario is deleted from Stylus Studio.

## Composing SQL/XML in Stylus Studio

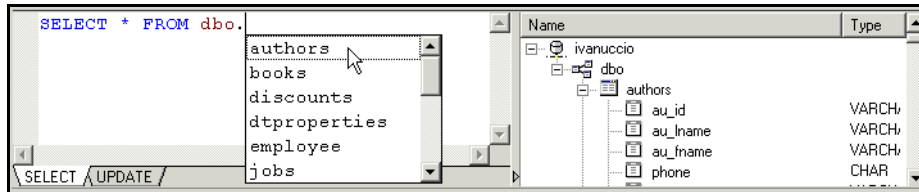
There are two ways to compose SQL/XML in Stylus Studio:

- Manually, typing SQL/XML on the **SELECT** and **UPDATE** tabs
- Automatically, using drag-and-drop to help you compose some or all of the SQL/XML

You can enter any valid SQL/XML statements in the SQL/XML editor's **SELECT** and **UPDATE** tabs, though only those statements for which you have permissions will be executed.

## Manually Entering SQL/XML

When you type a qualified name in the **SELECT** or **UPDATE** tabs of the editor, Stylus Studio's auto-completion feature displays a list of valid names. For example, when you type the period in this statement, `SELECT * FROM dbo.`, Stylus Studio displays the following:



**Figure 346. Example of Stylus Studio's Auto-Completion**

To finish the statement, select the object name from the drop-down list and press Enter (or just double-click).

## Using Drag-and-Drop

When you use drag-and-drop to compose SQL/XML, you start by selecting an object (a table or a column, for example) from the database schema tree and dragging it to the SQL/XML editor. When you drop the object (by releasing the mouse button) on the editor, Stylus Studio displays a shortcut menu of choices for that object. These include

- The object name, in both unqualified and qualified formats (authors and dbo.authors, for example). The format you choose depends on the database you are using – some databases require qualified names, for example. You might want to use this feature to add object names to an SQL/XML statement without typing them.
- A complete SELECT statement (if you are on the editor's **SELECT** tab); Stylus Studio uses the object name you selected to complete the statement.
- A complete INSERT, UPDATE, or SELECT statement (if you are on the editor's **UPDATE** tab); Stylus Studio uses the object name you selected to complete the statement.

### Example

Following is an example of the INSERT statement Stylus Studio creates using the publishers table:

```
INSERT xml_document(?)
INTO dbo.publishers (pub_id, pub_name, city, state, country)
xml_row_pattern('/root/row')
VALUES(
  xml_xpath('pub_id/text()', 'CHAR'),
  xml_xpath('pub_name/text()', 'VARCHAR'),
  xml_xpath('city/text()', 'VARCHAR'),
  xml_xpath('state/text()', 'CHAR'),
  xml_xpath('country/text()', 'VARCHAR'))
```

### How Relational Data is Translated to XML

Consider the following illustration of an excerpt from the authors table in the Microsoft SQL Server pubs database. This illustration shows only the first six columns of the table (and only the first few records); the state, zip, and contract columns have been omitted for clarity:

<i>au_id</i>	<i>au_lname</i>	<i>au_fname</i>	<i>phone</i>	<i>address</i>	<i>city</i>
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland
238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley
267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose

As you can see, the authors table has columns for author ID (*au\_id*), author's last name (*au\_lname*), author's first name (*au\_fname*), and so on.

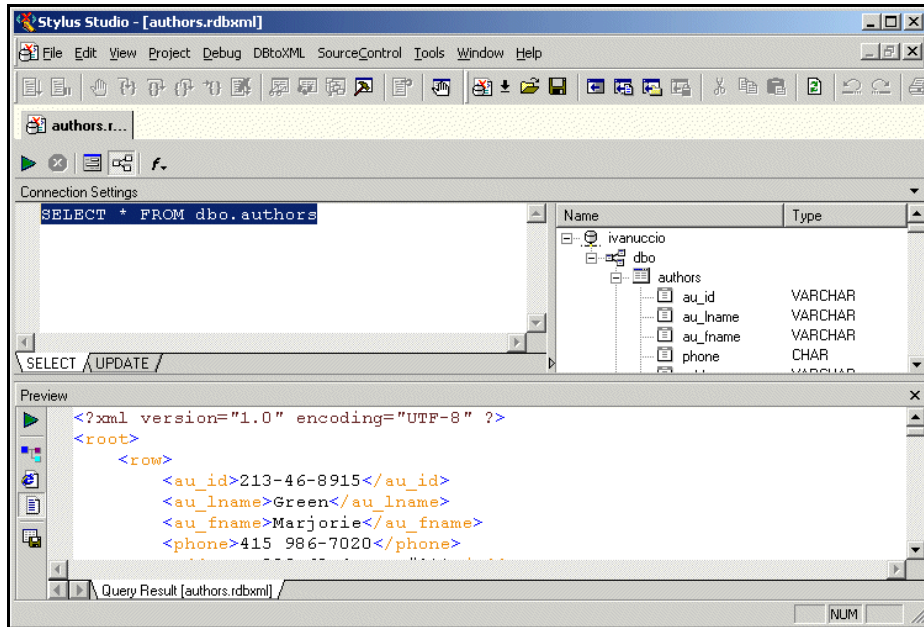
If we write a SELECT statement (SELECT \* FROM dbo.authors), Stylus Studio returns the following XML document:

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <row>
    <au_id>213-46-8915</au_id>
    <au_lname>Green</au_lname>
    <au_fname>Marjorie</au_fname>
    <phone>415 986-7020</phone>
    <address>309 63rd St. #411</address>
    <city>Oakland</city>
    <state>CA</state>
    <zip>94618</zip>
    <contract>true</contract>
  </row>
  <row>
    <au_id>238-95-7766</au_id>
    <au_lname>Carson</au_lname>
    <au_fname>Cheryl</au_fname>
    <phone>415 548-7723</phone>
    <address>589 Darwin Ln.</address>
    <city>Berkeley</city>
    <state>CA</state>
    <zip>94705</zip>
    <contract>true</contract>
  </row>
  <row>
    <au_id>267-41-2394</au_id>
    <au_lname>O'Leary</au_lname>
    <au_fname>Michael</au_fname>
    <phone>408 286-2428</phone>
    <address>22 Cleveland Av. #14</address>
    <city>San Jose</city>
    <state>CA</state>
    <zip>95128</zip>
    <contract>true</contract>
  </row>
</root>
```

Notice that each record in the table (that is, each author) is rendered as a separate `<row>` element. Similarly, each column is rendered as a subelement of `<row>`, taking the column name (`au_id`, `au_lname`, and so on) as its own (`<au_id>`, `<au_lname>`, and so on). The document's root element is given the name `<root>`.

## Working with Relational Data as XML

As shown in the example in the preceding section, an SQL/XML query is returned as XML. The XML is displayed in the **Preview** window in the DB-to-XML data source editor, as shown in [Figure 347](#).




**Figure 347. SQL/XML Query Returned as XML**

You can work with an SQL/XML query result as XML and, optionally, update the relational database using changes to the XML file and the SQL/XML statements defined on the **UPDATE** tab of the DB-to-XML data source editor.


## Understanding SELECT and UPDATE

As described previously, you define SQL/XML **SELECT** statements on the **SELECT** tab; you define **INSERT** and **UPDATE** statements on the **UPDATE** tab. The DB-to-XML data source simply presents the SQL/XML to the database. The database executes the SQL/XML to the best of its ability, based on the statements you have defined and the

permissions of the user associated with the DB-to-XML data source. If the user has read-only permissions, for example, any INSERT or UPDATE statement will fail.

**Tip** The **Output Window** can help you troubleshoot database problems. To display the **Output Window**, click the **Output Window** button ()

## When Statements are Executed

Statements on the **SELECT** tab are executed when you click the **Execute Query** button () . Statements on the **UPDATE** tab, on the other hand, are executed only when an XML version of the DB-to-XML data source document is saved. (The **Execute Query** button is disabled when the **UPDATE** tab is active.)

## Example

This example shows how to define a DB-to-XML data source, and how DB-to-XML data sources interact with a relational database.

Consider a DB-to-XML data source that performs a simple SELECT and INSERT.

## The SELECT Statement

The SELECT statement is defined on the **SELECT** tab of the DB-to-XML editor:

```
SELECT * FROM dbo.authors
```

This SELECT statement returns every author record from the `dbo.authors` table. Thus, the XML will have one `<row>` element for each author record in the table. There are currently 25 author records in the `dbo.authors` table.

### The INSERT Statement

The INSERT statement is defined on the **UPDATE** tab of the DB-to-XML data source editor:

```
INSERT xml_document(?)
INTO dbo.authors
(au_id,au_lname,au_fname,phone,address,city,state,zip,contract)
xml_row_pattern('/root/row')
VALUES(
  xml_xpath('au_id/text()','VARCHAR'),
  xml_xpath('au_lname/text()','VARCHAR'),
  xml_xpath('au_fname/text()','VARCHAR'),
  xml_xpath('phone/text()','CHAR'),
  xml_xpath('address/text()','VARCHAR'),
  xml_xpath('city/text()','VARCHAR'),
  xml_xpath('state/text()','CHAR'),
  xml_xpath('zip/text()','CHAR'),
  xml_xpath('contract/text()','BIT'))
```

This INSERT statement attempts to insert the entire XML document content into table `dbo.authors`, matching the document contents using `xml_row_pattern('/root/row')`.

### Saving the File

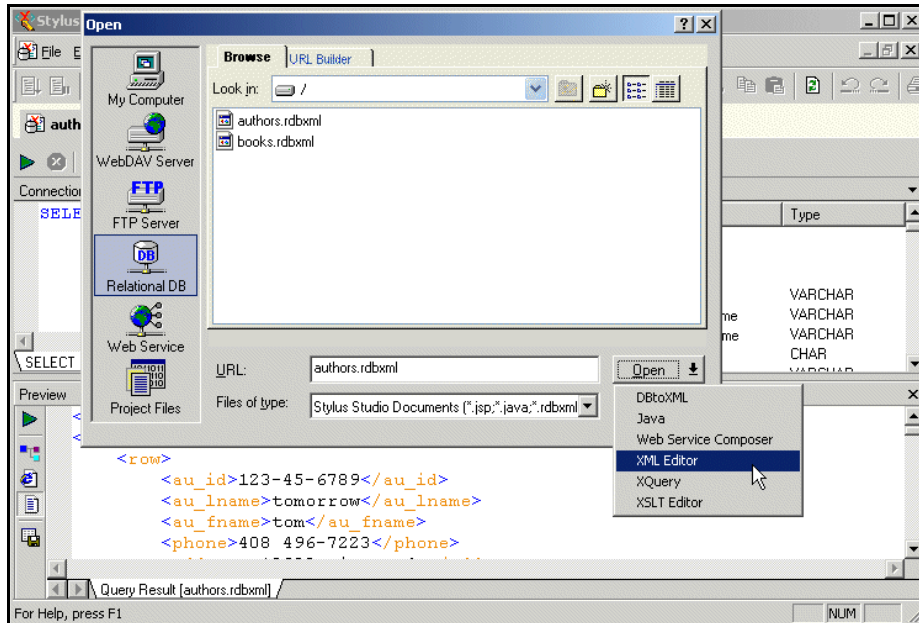
After executing the query from the DB-to-XML data source editor, we examine the query results, which are rendered as XML in the **Preview** window. (See [Figure 347 on page 784](#), for an example of how SQL/XML query results are displayed in Stylus Studio.) Checking the results allows us to verify that the table we are querying is providing us with the data we expect.

Once we have verified the query result, we save the `.rdxml` file. Although this file contains both SELECT and UPDATE statements, it is important to note that the SQL/XML defined on the **UPDATE** tab has not been executed at this point.



## Opening the .rdbxml as XML

Next, we open the .rdbxml file using the XML editor, as shown here:



**Figure 348. Opening a DB-to-XML Data Source as XML**

When the DB-to-XML data source is opened in the XML editor, it looks just like any other XML document – it displays the XML returned when we executed the SQL/XML query. Other data source attributes (connection settings and the SQL/XML statements themselves) are not accessible or visible when a data source is opened as XML, but they are present as metadata.

### Updating the Data in the Database

To update the data from the authors table in the database, we modify the XML in the .rdbxml file, adding the following record (a new <row> element):

```
<row>
  <au_id>781-23-4956</au_id>
  <au_lname>Black</au_lname>
  <au_fname>Frank</au_fname>
  <phone>301 282-7361</phone>
  <address>23 Bishop St.</address>
  <city>Baltimore</city>
  <state>MD</state>
  <zip>21281</zip>
  <contract>>false</contract>
</row>
```

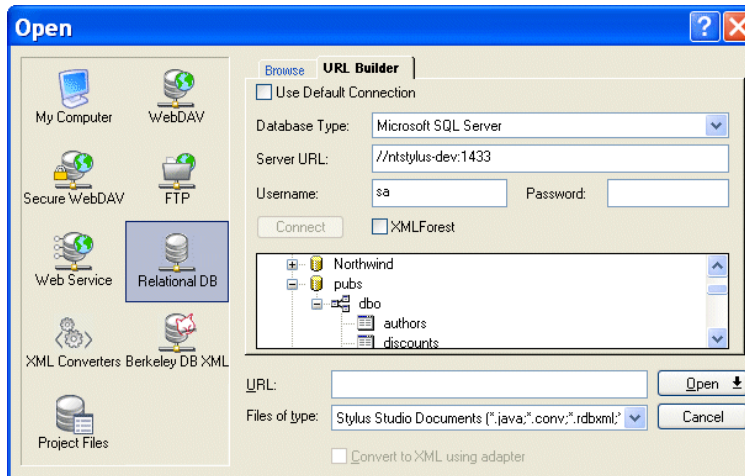
The document now contains a total of 26 <row> elements – the 25 that were read from the database, and the one which we just added. If we save this document as it is currently written, the INSERT statement we defined in the DB-to-XML data source will attempt to insert the *entire* document back into the database – all 26 <row> elements.

Similarly, if we deleted all existing <row> elements from the document, and saved the .rdbxml file with only the new record we created (the <row> element for author Frank Black), only that record would be added to table dbo.authors.

**Note** How the INSERT operation behaves – whether or not duplicate records are created on the database, for example – is determined by the controls for data integrity established by the database administrator.

## Using the URL Builder

The URL Builder utility lets you connect to a relational database server, select a table or view, and render that object as an XML document. You can use this feature, implemented on the **URL Builder** tab of the **Open** dialog box, to open relational data as an XML document anywhere in Stylus Studio – in the XML Editor, the XQuery and XSLT Mappers, and so on.



**Figure 349. URL Builder Tab of the Open Dialog Box**

In this section

This section covers the following topics:

- [“URL Builder Compared to DB-to-XML Data Sources”](#) on page 789
- [“When to Use”](#) on page 790
- [“Connection Settings”](#) on page 790
- [“How to Open a Table or View as an XML Document”](#) on page 792

### URL Builder Compared to DB-to-XML Data Sources

Using the URL Builder is similar to using DB-to-XML data sources with the following notable exceptions:

- The URL Builder does not expose SQL/XML. You cannot write SQL/XML queries using the URL Builder.

- The URL Builder allows you to select only a single table or view from the relational database server. You cannot select multiple tables, views, or individual rows using the URL Builder.
- The URL Builder does not allow you to save XML documents back to the relational database server.

### When to Use

Use the URL Builder any time you want to quickly render an entire table or view as XML. Any time you need more control – writing your own SQL/XML or updating the relational database, for example – use a DB-to-XML data source. See [“Overview of DB-to-XML Data Sources”](#) on page 766 for more information.

### Connection Settings

Connection settings for the URL Builder are the same as those used to define a DB-to-XML data source:

- Database type
- Server URL
- Username
- Password

You can specify a new set of connection properties, or you can use a default connection by clicking the **Use Default Connection** check box. See [“Specifying Connection Settings”](#) on page 773 for more information.

### XMLForest

Stylus Studio supports *XMLForest*, an SQL/XML function that produces a sequence of XML elements based on the rows in a relational table. You can specify whether or not you want to use XMLForest when using the URL Builder to access a relational table as XML.

## Example

Consider a simple BOOKS table with a TITLE row. If you accessed the BOOKS table using the URL Builder alone, the resulting XML would look like this:

```
<BOOKS>
  <row>
    <TITLE>A History of Moto GP</TITLE>
  </row>
  <row>
    <TITLE>Motorcycles: Art in Design</TITLE>
  </row>
</BOOKS>
```

Stylus Studio creates the <BOOKS> element as the root, with separate <row> elements for each row in the table, one row for each title.

When you use XMLForest, the resulting XML is not well-formed – it has no root element, and the data is represented as a sequence of <BOOKS> elements instead of as individual rows, as shown here:

```
<BOOKS>
  <TITLE>A History of Moto GP</TITLE>
</BOOKS>
<BOOKS>
  <TITLE>Motorcycles: Art in Design</TITLE>
</BOOKS>
```

## Sequence Elements

To allow you to work with sequences formed by XMLForest in Stylus Studio, the URL Builder adds a *sequence element* to the XML. The sequence element acts as a root element, ensuring that the XML is well-formed. The sequence element takes the following form:

```
<stylus:sequence xmlns:stylus="http://www.stylusstudio.com/xquery">
...
</stylus:sequence xmlns:stylus="http://www.stylusstudio.com/xquery">
```

In most cases, the sequence element is ignored for display purposes in Stylus Studio.

## How to Open a Table or View as an XML Document

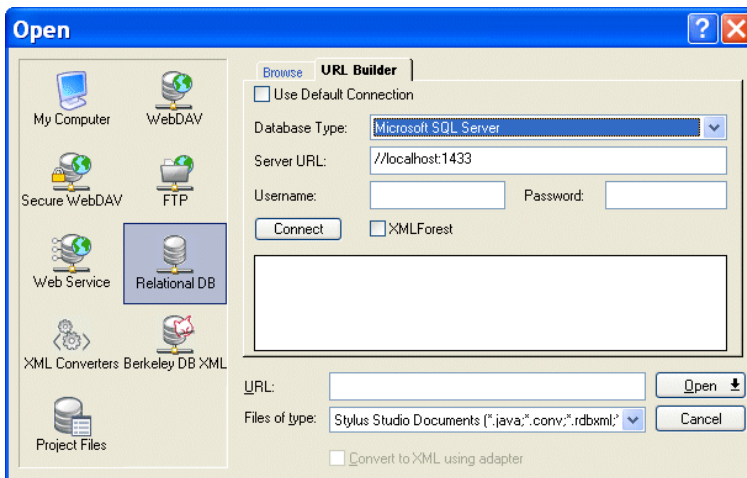
◆ **To open a table as an XML document:**

1. Display the **Open** dialog box.
2. Click the Relational DB icon.



Two new tabs appear: **Browse** and **URL Builder**.

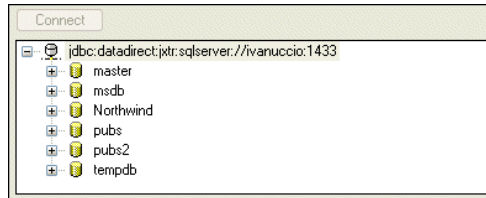
3. Click the **URL Builder** tab.



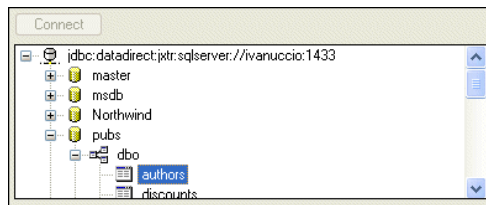
4. Complete the connection settings as described in “[Specifying Connection Settings](#)” on page 773.

5. Click the **Connect** button.

The databases for the server you connected to appear in the **URL Builder** tab. For example:

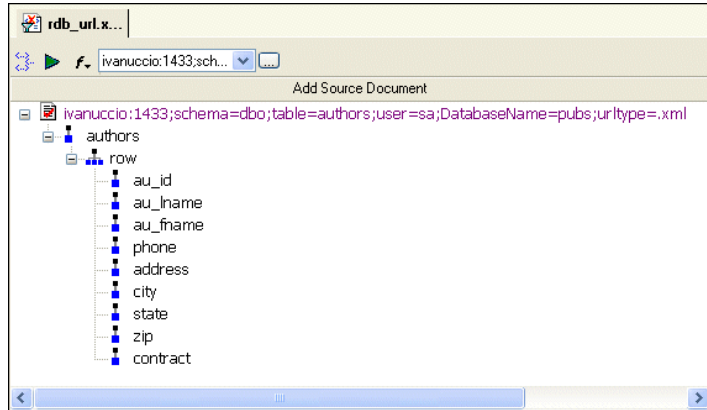


6. Expand the tree diagram to locate the table or view you want to open as an XML document. For example:



7. Click the **Open** button.

Stylus Studio opens the table or view as an XML document in the editor you were using when you displayed the **Open** dialog box. In this example, the authors table was opened as a source document in the XQuery Mapper.



Notice that the document name contains the connection settings used to define the URL in [Step 4](#).



## Chapter 13   **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 and custom file systems.



Custom file systems and integration with Berkeley DB XML are available only in Stylus Studio XML Professional Edition.

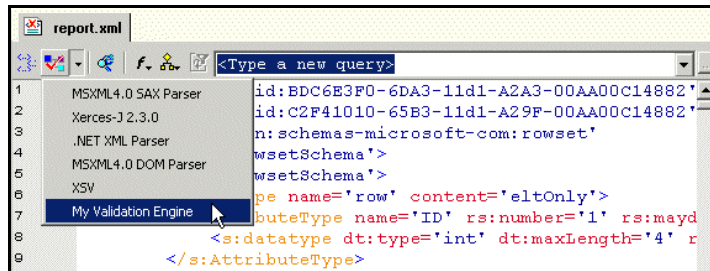
This chapter discusses the following topics:

- [“Custom XML Validation Engines”](#) on page 795
- [“Custom Document Wizards”](#) on page 801
- [“Stylus Studio File System Java API”](#) on page 811
- [“Using Stylus Studio with Berkeley DB XML”](#) on page 817

### **Custom XML Validation Engines**

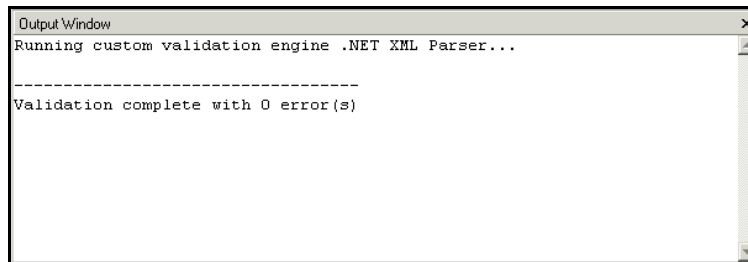
Stylus Studio supports several XML validation engines, including the MSXML4.0 SAX Parser, Xerses-J 2.3.0, and .NET XML. The custom validation engine feature lets you register your own XML validation engine with Stylus Studio. Custom validation engines

are added to the **Validate Document** drop-down list in the XML Editor once you register them with Stylus Studio, as shown in [Figure 350](#).



**Figure 350. 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 351. 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 802.

## 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 797
- [How to Configure a Custom Validation Engine](#) on page 800

### 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.

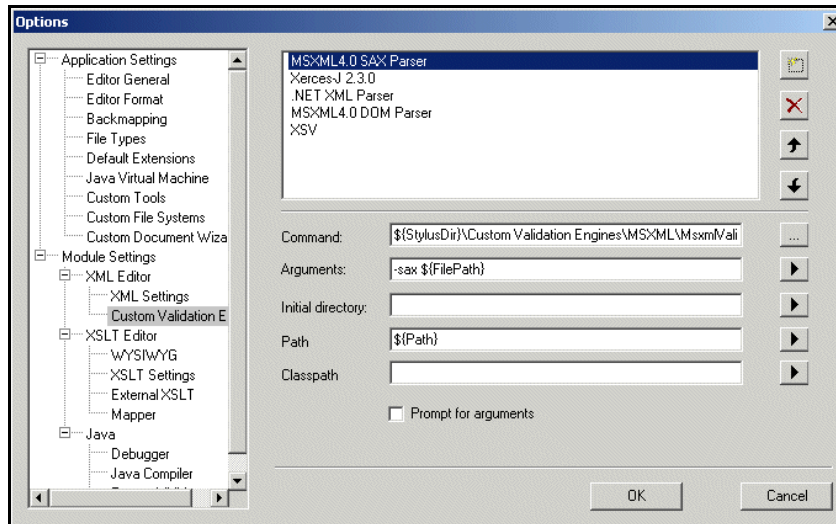


Figure 352. Custom Validation Engines Page

### 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 353. 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 354. 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 you begin

Before performing this procedure, you should be familiar with the information in [“The Custom Validation Engines Page”](#) on page 797.

### ◆ To configure a custom validation engine:

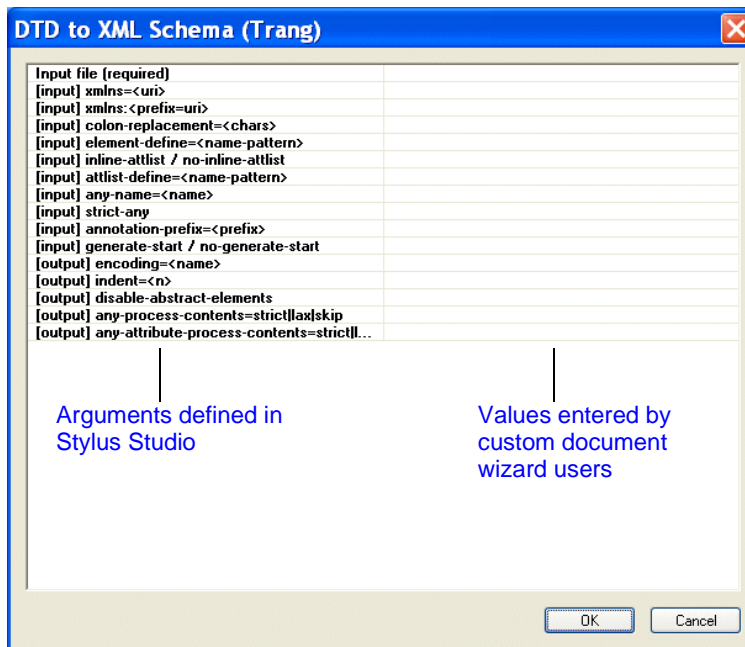
1. Display the **Custom Validation Engines** page of the **Options** dialog box. See [“How to display”](#) on page 798 if you need help with this step.
2. Click the **New** button and enter a name for the custom validation engine. Remember that this value is displayed in the **Validate Document** drop-down list in the XML Editor.
3. Specify the command line any required arguments. See [“Command”](#) on page 799 and [“Arguments”](#) on page 799 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.

Example

An example of a custom document wizard is the DTD to XML Schema (Trang) document wizard shipped with Stylus Studio.



**Figure 355. 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 802.

## Configuring a Custom Document Wizard

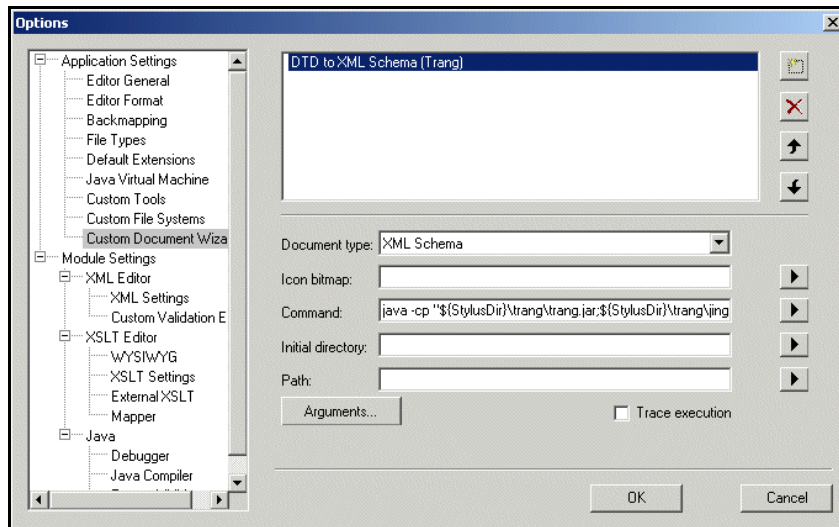
This section provides information and procedures for configuring a custom document wizard. It covers the following topics:

- [“The Custom Document Wizards Page”](#) on page 803
- [“Defining Arguments”](#) on page 807
- [“How to Configure a Custom Document Wizard”](#) on page 811



## 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 356. 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 .

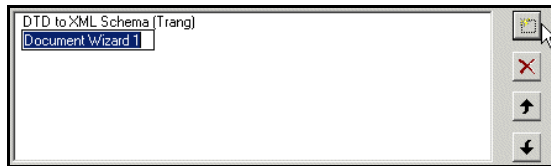
Predefined macros include

- `${StylusDir}`, which indicates that the path you are specifying is relative to the Stylus Studio installation directory.
- `${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.



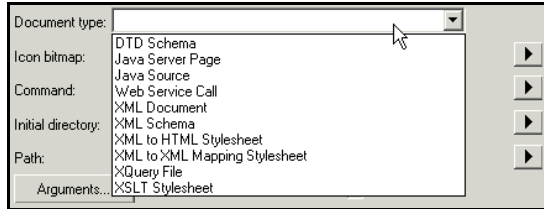
**Figure 357. Specifying a Custom Document Wizard Name**

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 358. 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 807 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

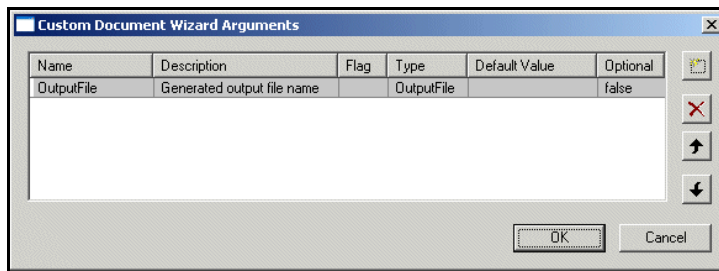
- Browse for and select a file you want to specify.
- Insert the `${StylusDir}` macro.
- Insert the `${PATH}` macro.

## 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 359. 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 “[Example](#)” on page 801 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 79](#) summarizes valid values for the **Type** field and describes possible values for those types

**Table 79. 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 <b>Flag</b> attribute in the command line. No value other than the <b>Flag</b> value appears in the command line for Boolean arguments. If the value is false, the associated <b>Flag</b> value does not appear in the command line.  If you set <b>Type</b> to boolean, you must specify the argument’s <b>Flag</b> attribute.
InputFile	The value for an InputFile argument is a URL that the custom document wizard user enters or selects by clicking the <b>Browse</b> 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.

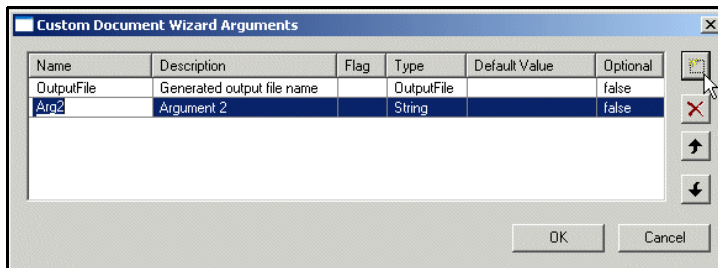
- Optional. Whether or not the argument is optional. Valid values for this field are true or false.

### How to define an argument

#### ◆ To define an argument:

1. Display the **Custom Document Wizard Arguments** dialog box. See “[How to display](#)” on page 807 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 360. 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 you begin

Before performing this procedure, you should be familiar with the information in [“The Custom Document Wizards Page”](#) on page 803 and [“Defining Arguments”](#) on page 807.

◆ **To configure a custom document wizard:**

1. Display the **Custom Document Wizards** page of the **Options** dialog box. See [“How to display”](#) on page 803 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 807 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 805 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**.

## Stylus Studio File System Java API



The Stylus Studio File System Java API is available only in Stylus Studio XML Professional Edition.

The Stylus Studio File System Java API is a set of interfaces that allows you to

- Create and register your own custom file systems
- Programmatically call an adapter to convert a file from one format to another using Stylus Studio’s built-in converters, or converters you define using the Convert to XML module

This section provides an overview of the Stylus Studio File System Java API and describes how to create and register a custom file system. It assumes knowledge of Java programming principles and procedures.

In this section

This section covers the following topics:

- [“Overview”](#) on page 812
- [“Custom File Systems”](#) on page 813
- [“File System Interfaces”](#) on page 814
- [“Registering a Custom File System”](#) on page 815

For more information

To learn how to use the Stylus Studio File System Java API to programmatically call an adapter, see [“Invoking an Adapter Programmatically”](#) on page 287.

## Overview

The Stylus Studio File System Java API consists of the following classes, summarized in [Table 80](#).

**Table 80. File System Java API Classes**

<b><i>Class</i></b>	<b><i>Description</i></b>
<i>StylusFile</i>	Public interface that provides an abstract representation of a file in a custom file system.
<i>StylusFileFactory</i>	Public final class that defines a factory API that enables applications to read and write data streams using any file system that implements the <i>StylusFile</i> interface.
<i>StylusFileHelpers</i>	Public class of static helpers for use with <i>StylusFile</i> and <i>StylusFileSystem</i> interfaces and the <i>StylusFile</i> class.
<i>StylusFileSystem</i>	Public interface that represents a custom file system, enabling you to connect to and and browse files on that file system.
<i>StylusFileSystemOptions</i>	Public interface that provides optional methods

These files are packaged in `CustomFileSystem.jar`, which is in the Stylus Studio `\bin` directory.

## Javadoc for the Stylus Studio File System Java API

Javadoc for the Stylus Studio File System Java API is installed with Stylus Studio. It is available in the Stylus Studio `/doc/Javadoc` directory. Open [index.html](#) to get started. The Javadoc is also available on the Stylus Studio Web site, <http://www.StylusStudio.com>.

## Custom File Systems

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.

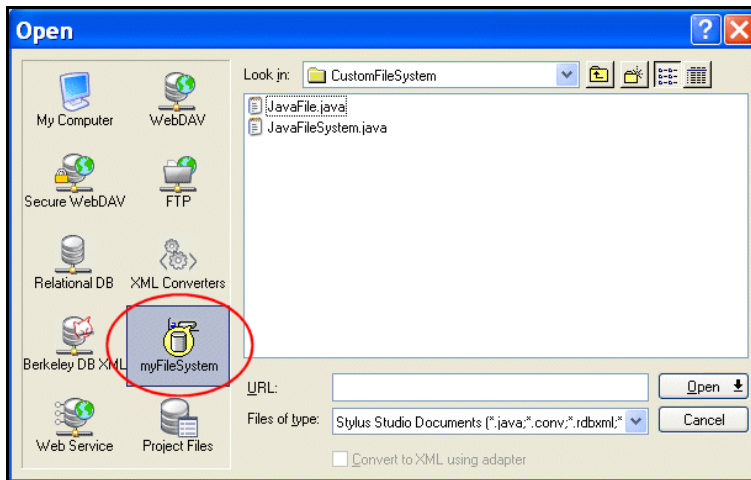
**Tip** The Stylus Studio interface to Sleepycat Software’s Berkeley DB XML was implemented using the File System Java API to create a custom file system.

## 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, create the Java wrapper class for your data source. See “[File System Interfaces](#)” on page 814 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 361. See “Registering a Custom File System” on page 815 for more information on this step.



**Figure 361. 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`, and
- `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

Before you begin

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.

In this section

This section covers the following topics:

- “The Custom File Systems Page” on page 815
- “How to Display” on page 816
- “Fields” on page 816
- “How to Register a Custom File System” on page 817

### The Custom File Systems Page

You use the **Custom File Systems** page of the **Options** dialog box to register a custom file system.

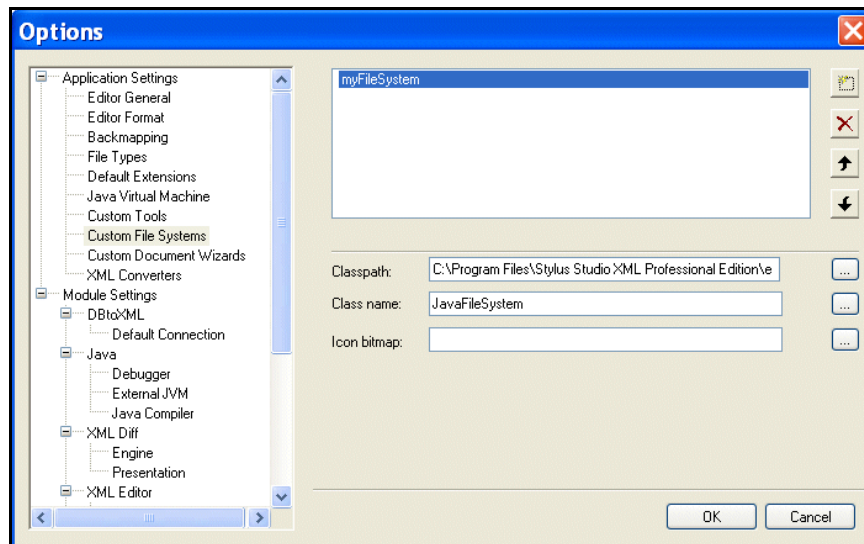


Figure 362. Options Dialog Box for Custom File Systems

## 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 363. Naming a New Custom File System**

You should replace the default name (**FileSystem1**, 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 816 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**.

## Using Stylus Studio with Berkeley DB XML



Integration with the Berkeley DB XML file system is available only in Stylus Studio XML Professional Edition.

Stylus Studio allows you to access XML documents stored on Sleepycat Software’s Berkeley DB XML database and to write documents back to the database. This section describes prerequisites, as well as the configuration and procedural information you need to get started.

This section covers the following topics:

- “[Overview](#)” on page 818
- “[Prerequisites](#)” on page 819
- “[Configuring the BerkeleyDBXML.xml File](#)” on page 820
- “[Naming Files](#)” on page 823
- “[Opening a File Stored on Berkeley DB XML](#)” on page 824
- “[Saving a File Back to Berkeley DB XML](#)” on page 824

- “Creating Containers with Stylus Studio” on page 826
- “Usage Tips” on page 827

### Overview

Stylus Studio can read and write well-formed XML documents (.xml, .xsl, and so on) to/from Berkeley DB XML container files (containers). Containers typically have a .dbxml extension. In Stylus Studio, you reference external files using a URL. The URL used to access Berkeley DB XML files is `dbxml:///container_pathname/filename.xml`:

- `dbxml` is the prefix that specifies that files will be read from Berkeley DB XML
- `container_pathname` is the pathname of a Berkeley DB XML container
- `filename.xml` is the name of the file you wish to access within that container

For example, to access one of the sample files provided with Berkeley DB XML, you might enter the following in the **URL** field of the Stylus Studio **Open** dialog box:  
`dbxml:///C:/samples/simpleExampleData.dbxml/Allspice.xml`

Note that the container name (`C:/samples/simpleExampleData.dbxml` in the example) appears in the **URL** field as if it were a directory name. This is because Stylus Studio treats Berkeley DB XML containers very much like directories, and the XML documents stored within a container are treated like files within a directory.

You can save files back to the same container from which they were read, to a new container, or to some other file system (your local machine, for example).

**Note** You can use a Berkeley DB XML URL to open an XML document anywhere you can specify URLs in Stylus Studio. For example, you can use the Berkeley DB XML URL to specify the source document for XQuery Mapper.

### Berkeley DB XML Support

Stylus Studio supports Berkeley DB XML Version 1.2.1, Version 2.0.7, and Version 2.0.9.



## Usage Summary

The process for using Stylus Studio with Berkeley DB XML consists of the following steps:

1. Make Berkeley DB XML .jar and .dll files accessible to Stylus Studio. See [“Prerequisites”](#) on page 819 for more information on this step.
2. Configure the BerkeleyDBXML.xml file as described in [“Configuring the BerkeleyDBXML.xml File”](#) on page 820.
3. Open Berkeley DB XML files in Stylus Studio as described in [“Opening a File Stored on Berkeley DB XML”](#) on page 824.
4. Optionally, save your work back to Berkeley DB XML as described in [“Saving a File Back to Berkeley DB XML”](#) on page 824.

Each of these steps is described in greater detail in the sections that follow.

## Prerequisites

In order for Stylus Studio to access files stored on the Berkeley DB XML database, it must be able to access the .jar and .dll files shown in [Table 81](#). Note that specific files vary based on the Berkeley DB XML version you are using, as shown in the following table.

**Table 81. Sleepycat Files Required for Stylus Studio Access**

<i>Berkeley DB XML V1.2.1</i>	<i>Berkeley DB XML V2.0.7 and 2.0.9</i>
db.jar (version-specific)	db.jar (version-specific)
dbxml.jar (version-specific)	dbxml.jar (version-specific)
libdb42.dll	libdb43.dll
libdb_java42.dll	libdb_java43.dll
libdbxml12.dll	libdbxml20.dll
libdbxml_java12.dll	libdbxml_java20.dll
Pathan.dll	Pathan_7.1.dll
Xerces-c_2_4_0.dll	Xerces-c_2_4_0.dll

These files are supplied by Sleepycat software – either as part of a pre-built binary distribution of Berkeley DB XML, or built by you if you received a source code

distribution from Sleepycat. These files are not provided as part of your Stylus Studio installation.

Once you have acquired these files, you must make them accessible to Stylus Studio. You do this by modifying your CLASSPATH and PATH environment variables as follows:

- The Java CLASSPATH must specify the location of the .jar files
- The PATH must specify the location of the .dll files

These environment variables must be established before you start Stylus Studio. You cannot set them within Stylus Studio.

**Tip** Stylus Studio supports Version 1.2.1, Version 2.0.7, and Version 2.0.9, but not in the same session. To switch Berkeley DB XML versions, change the CLASSPATH and PATH environment variables as needed before starting Stylus Studio.

### Saving Files to Berkeley DB XML

In order to save files to Berkeley DB XML, you must first edit the settings in a configuration file. See “[Configuring the BerkeleyDBXML.xml File](#)” on page 820.

## Configuring the BerkeleyDBXML.xml File

You need to configure the BerkeleyDBXML.xml file in order to be able to save XML documents back to a Berkeley DB XML container. Until you configure the BerkeleyDBXML.xml file as described in this section, any files you read from Berkeley DB XML are marked read-only and cannot be written back to Berkeley DB XML.

### The Berkeley Database Environment

The Berkeley database has the concept of a *database environment*. The database environment is typically a directory that contains the log and lock files that provide support for transaction processing. This directory may also contain databases – files that contain user data. The containers in which Berkeley DB XML stores XML documents are examples of Berkeley databases and typically, though not always, have a .dbxml extension.

To be able to determine what database environments exist in Berkeley DB XML, and which ones are associated with which containers, you need to

- Modify the BerkeleyDBXML.xml configuration file
- Validate your changes using the BerkeleyDBXML.xsd XML Schema

These files are installed in the `\bin\lib\sleepycat` directory where you installed Stylus Studio (`c:\Program Files\Stylus Studio XML Professional Edition\bin\lib\sleepycat`, for example).

## The BerkeleyDBXML.xml Configuration File

The BerkeleyDBXML.xml configuration file contains elements that allow you to specify Berkeley DB XML database environments and directories:

- `<dbenv>` – The pathname of a database environment.
- `<dbdir>` – A directory within that database environment that stores the containers you wish to access.

A finished BerkeleyDBXML.xml configuration file might look like this:

```
<dbxml>
  <dbenv path="c:\dbenv1">
    <dbdir path="d:\datadir1"/>
    <dbdir path="e:\datadir2"/>
  </dbenv>
  <dbenv path="c:\testenv">
    <dbdir path="c:\testdata"/>
  </dbenv>
  <dbenv path="c:\testenv2"/>
</dbxml>
```

### Omitting `<dbdir>` and `<dbenv>` elements

In some cases, it is not necessary to have any `<dbdir>` elements, as is illustrated by the `<dbenv path="c:\testenv2"/>` element in the example BerkeleyDBXML.xml configuration file. You can omit the `<dbdir>` element

- If it has the same path as its `<dbenv>` parent.
- If you have a `DB_CONFIG` file in a database environment – Stylus Studio reads the `set_data_dir` entries in that file. (See the Berkeley DB XML API documentation for more information on the `DB_CONFIG` file.)

If you use the `DB_HOME` environment variable to locate your database environment and all its data directories are listed in the `DB_CONFIG` file for that database environment, you can omit that database environment from the BerkeleyDBXML.xml configuration file. Indeed, if that is your only database environment, you do not need the BerkeleyDBXML.xml configuration file at all.

### Specifying <dbdir> pathnames


Pathnames you specify for <dbdir> elements can be absolute or relative. Relative pathnames are relative to the enclosing <dbenv> pathname, not to your current working directory.

### The validate attribute

In Berkeley DB XML Version 2.0.7 and 2.0.9, the <dbxml>, <dbenv>, and <dbdir> elements have a `validate` attribute that allows Sleepycat to validate XML documents before saving them to the Berkeley DB XML file system. See “[Validating Documents](#)” on page 825 for more information on this topic.

## How to Configure BerkeleyDBXML.xml

### ◆ To configure BerkeleyDBXML.xml:

1. Open the BerkeleyDBXML.xml configuration file in Stylus Studio.
2. Specify the <dbenv> and <dbdir> elements as described in “[The BerkeleyDBXML.xml Configuration File](#)” on page 821.
3. Click **Validate Document** .
4. If the BerkeleyDBXML.xml configuration file is valid, save your changes.  
You are ready to use Berkeley DB XML with Stylus Studio.

## Open the Output Window

The first time you perform an operation on the Berkeley DB XML, Stylus Studio tries to process the BerkeleyDBXML.xml file. As the file is processed, messages are written to the Stylus Studio **Output** window. These messages identify which database environments and directories are being processed. This can be a useful way to verify which directories Stylus Studio is associating with which database environment and identify when you might need to make a change in the BerkeleyDBXML.xml file.

## Naming Files

File-naming conventions vary between Berkeley DB XML V1.2 and V2.0.

**Table 82. Sleepycat Files Required for Stylus Studio Access**

<i>Berkeley DB XML V1.2.1</i>	<i>Berkeley DB XML V2.0.7 and 2.0.9</i>
File names are not required	File names are required
File names do not have to be unique; Berkeley DB XML does, however, assign a unique ID number to each file when it is first saved to the container.	File names must be unique

Stylus Studio creates a unique pseudo-name for each named file in a container. Files without a name (supported only in Berkeley DB XML Version 1.2.1) are invisible to Stylus Studio. The pseudo-name consists of the file's name, concatenated with a pound sign (#) and the file ID assigned by Berkeley DB XML.

### Example – Berkeley DB XML V 1.2.1

Consider the following files in a container as seen by Berkeley DB XML V 1.2.1:

- flotsam.xml
- flotsam.xml
- jetsam.xml
- [a file without a name]

When you display this container in the Stylus Studio **Open** dialog box, only these files appear:

- flotsam#1.xml
- flotsam#2.xml
- jetsam.xml

Note that only duplicate file names are assigned a numeric ID. The fourth, unnamed, file, is not visible to Stylus Studio.

## Opening a File Stored on Berkeley DB XML

You can open files stored on Berkeley DB XML in one of two ways:

- Using the **Open** dialog box, as described here.
- By entering the Berkeley DB XML URL for the document you want to open (dbxml:///C:/samples/simpleExampleData.dbxml/Allspice.xml, for example) in any field in Stylus Studio that allows you to specify a resource.


**Note** You can access only those files stored in containers. You cannot open standalone XML documents in Berkeley DB XML.

### ◆ To open a file stored on Berkeley DB XML:

1. In the File Explorer, click the **Berkeley DB XML** icon.  
*Alternative:* From the Stylus Studio menu bar, select **File > Open**, and then click the **Berkeley DB XML** icon displayed in the **Open** dialog box.
2. Navigate the file system and locate the Berkeley DB XML database that contains the file you want to open.
3. Select the file you want to open.
4. Click the **Open** button.

## Saving a File Back to Berkeley DB XML

You save a file to Berkeley DB XML as you would save a file to any other file system – click **File > Save**. If the document is not well-formed, Berkeley DB XML will not write it to the container.

**Tip** Use the Stylus Studio well-formedness checker  to check your XML documents before saving them to Berkeley DB XML.

## Saving and Containers

By default, **File > Save** saves the file back to the same container from which it was read. Additionally, you can

- Save the file to any container you choose, provided that you have specified it (and its database environment) in the BerkeleyDBXML.xml file.
- Create a new container on the fly by specifying it in the URL. See [“Creating Containers with Stylus Studio”](#) on page 826.

In any case, in order to save any work back to Berkeley DB XML, you must have previously configured the BerkeleyDBXML.xml file. See [“Configuring the BerkeleyDBXML.xml File”](#) on page 820.

## Validating Documents

If you are using Berkeley DB XML Version 2.0.7 or 2.0.9, you can optionally have Sleepycat validate XML documents before saving them to the database. You do this using the `validate` attribute, which you can set in the `<dbxml>`, `<dbenv>`, and `<dbdir>` nodes in the BerkeleyDBXML.xml file, as shown in the following example:

```
<dbxml validate="true">
  <dbenv path="c:\dbenv1" validate="false">
    <dbdir path="d:\datadir1"/>
    <dbdir path="e:\datadir2" validate="true"/>
  </dbenv>
  <dbenv path="c:\testenv" validate="true">
    <dbdir path="c:\testdata"/>
  </dbenv>
</dbxml>
```

The syntax for the `validate` attribute is `validate=["true" | "false"]`. By default, this attribute is not specified and validation is not performed.

Validation occurs at the container level; each container, specified as a node in the BerkeleyDBXML.xml file, can have its own settings for the `validate` attribute. A child node inherits the `validate` attribute from its parent; similarly, you can override settings for child nodes, as shown in the following example:

```
<dbxml>
  <dbenv path="c:\dbenv1" validate="true">
    <dbdir path="d:\datadir1"/>
    <dbdir path="e:\datadir2" validate="false"/>
  </dbenv>
</dbxml>
```

Here, the `validate` setting of the `<dbenv>` node overrides the (implicit) default of `"false"` of its parent, `<dbxml>`. The first child of `<dbenv>` inherits the `validate="true"` setting, and its second child overrides it.

**Note** The `set_database_dir` entries read from the `DB_CONFIG` file do not contain any `validate` settings. To validate database directories established by the `DB_CONFIG` file, enter them in the BerkeleyDBXML.xml file.

### Schema requirements

In order for Sleepycat to validate an XML document, the XML document must have its schema defined in an internal DTD. If the document does not contain a DTD, or references an external XML Schema, the document is not validated and will be saved if it is well-formed.

### Changing settings

Each time Stylus Studio starts, it reads the BerkeleyDBXML.xml configuration file and uses its settings for the entire session. If you want to change the validate attribute for a container, you must exit Stylus Studio, change the BerkeleyDBXML.xml file, and then restart Stylus Studio.

**Tip** Remember to validate the BerkeleyDBXML.xml file in Stylus Studio any time you edit it.

## Creating Containers with Stylus Studio

Typically, you use Stylus Studio to access files in existing Berkeley DB XML containers. If you want, you can use Stylus Studio to create new containers. Stylus Studio creates a new container whenever

- You type a complete Berkeley DB XML URL in the **URL** field in the **Save** or **Save As** dialog box when saving a file from Stylus Studio. For example:  
dbxml:///C:/samples/myNewContainer.dbxml/newWork.xml
- The output file name given to the command line utility StylusXslt.exe names a container that does not exist. See [“Using Stylus Studio from the Command Line”](#) on page 147 for more information on this utility.

In order to create a new container:

- The container name must end in .dbxml.
- The container home must be in a known database environment, either as specified in the BerkeleyDBXML.xml configuration file, or DB\_HOME

### Node and Wholedoc Containers

Berkeley DB XML Version 2.0.7 and 2.0.9 support Node and Wholedoc containers. Berkeley DB XML Version 1.2.1 supports only Wholedoc containers. Containers created by Stylus Studio are always created as Wholedoc containers.



## Usage Tips

Following are some tips to help you work with Berkeley DB XML in Stylus Studio.

### Before Starting Stylus Studio

Before you start Stylus Studio, you must

- Set your PATH and CLASSPATH environment variables
- Configure the BerkeleyDBXML.xml file. If any errors occur while accessing Berkeley DB XML – if a database pathname in the configuration file was incorrectly specified, for example – you must correct the problem and then restart Stylus Studio.

**Note** Once Stylus Studio has successfully read the BerkeleyDBXML.xml file, it will not read it again until you restart Stylus Studio.

### Database Environment Recovery

Database environment recovery applications must have exclusive access to the database environment they are trying to restore. If you need to perform a database environment recovery, be sure to shut down Stylus Studio – and any other applications that might be using that database environment – prior to starting recovery procedures.



---

# Index

## A

- adapters
  - converting files using adapters 235
  - creating 278
  - for converting EDI 284
  - opening files with 281
- ancestor axis 643
- ancestor-or-self axis 647
- AND operator 655
- applying stylesheets
  - how it is done 323
  - Stylus Studio function key for 357
- attribute axis 645
- attributes
  - matching template 346
- automatic tag completion
  - Sense:X 9
  - Stylus Studio feature for 9
- axis syntax in queries 641

## B

- back-mapping
  - described 36
  - using in templates 485
  - XSLT processors that support 386
- backup copies of documents 233
- Berkeley DB XML
  - configuring the BerkeleyDBXML.xml file 820
  - creating containers in 826

- integration with Stylus Studio 817
  - opening files stored on 824
  - saving files to 824
- BerkeleyDBXML.xml file
  - configuring 820
- binary files
  - converting to XML 235
- BLOBs
  - querying 616
- bookmarks
  - setting 169
  - XQuery debugging and 728
- bookstore.xml 618
- boolean() function 656
- Booleans
  - converting operands to 656
  - expressions 655
  - functions 656
- breakpoints
  - in stylesheets 480
  - in XQuery documents 725

## C

- call stack
  - displaying the Call Stack window in the XQuery editor 727
- canonical XML
  - converting XML to canonical XML 229

- case sensitivity
  - Boolean operators and 655
  - queries and 632
- `ceiling()` function 661
- chaining stylesheets 148
- checking spelling 171
- child axis 642
- classpath
  - setting for a project 125
- ClearCase
  - using with Stylus Studio 129, 131
- command line
  - custom document wizard arguments 807
  - running Stylus Studio from 148
  - running XML diff 223
  - utilities 147
  - XML validation utility 150
  - XQuery utility 149
  - XSLT utility 148
- comma-separated files
  - see* CSV files
- `comment()` function 672
- comparing node sets 663
- comparing XML documents 195
  - merged view 208
  - text view 207
  - tree view 206
- `concat` function blocks
  - in XQuery 722
- `concat()` function 651
- concatenating strings 651
- conditions
  - expressing
    - in XQuery mapper 723
    - in XSLT 373
  - expressing in XSLT mapper 462
- containers
  - creating Berkeley DB XML containers in Stylus Studio 826
- `contains()` function 649
- context node 635
- context node set 635
- `.conv` files
  - default Stylus Studio module and 110
- Convert to XML
  - creating an adapter 278
  - field names
    - display of 243
    - specifying field names in XML output 260
  - video demonstration 235
- Convert to XML Editor
  - changing fonts in 246
  - display features for document pane 244
  - displaying grid lines in 246
  - displaying pattern matches in 245
  - displaying the ruler in 244
  - Go To dialog box 246
- converting EDI to XML 284
- converting files
  - using the Java API 287
- converting operands
  - to Booleans 656
  - to numbers 659
  - to strings 653
- `count()` function 678
- creating templates
  - example 348
  - how to 385
- creating XQuery
  - using the XQuery Mapper 700
- creating XSLT
  - using the XSLT Mapper 439
- CSV files
  - converting to XML
    - Convert to XML module 235
    - document wizard 156
  - formatting XML output
    - document wizard 157
- current context
  - number of nodes in 678
- current node
  - definition 635
- `current()` function 680
- custom document wizards
  - about 801
  - arguments 807
  - command line arguments 807
  - configuring 802
  - defining arguments 807

- document types for 805
- how to configure 811
- macros for 803
- naming 804
- registering 802
- specifying a command line template for 805
- Custom Document Wizards page in Options
  - dialog box 803
- custom file system
  - Javadoc for Java API 813
- custom file systems
  - Custom File Systems page 815
  - definition 813
  - Java API for 814
  - naming 816
  - registering with Stylus Studio 815
- custom validation engines for XML
  - about 795
  - configuring 797
  - macros for 798
  - naming 798
  - registering 796
- Custom Validation Engines page in Options
  - dialog box 797

## D

- data source
  - DB-to-XML data sources in Stylus Studio 766
- databases
  - accessing relational data from Stylus Studio 765
  - connecting
    - creating a default connection 775
  - connecting to 773
  - creating a default connection 775
  - DataDirect SequeLink and 766
  - integration with Berkeley DB XML 817
  - integration with Sleepycat Software 817
  - supported relational databases 766
  - using SQL/XML in Stylus Studio 780
  - using XMLForest with 790
- DataDirect SequeLink
  - Stylus Studio and 767
  - using with Stylus Studio 774
- DB-to-XML Data Source Editor
  - database schema pane 769
  - description 768
- DB-to-XML data source editor
  - SELECT and UPDATE tabs 768
- DB-to-XML data sources
  - composing SQL/XML for 780
  - creating 770
  - database connection settings 773
  - definition 766
  - example 785
  - how SELECT and UPDATE statements are executed 784
  - opening 771
  - saving 771
  - scenarios for 767
  - system requirements 766
  - video demonstration 765
- debugging
  - using bookmarks in the XQuery debugger 728
  - XSLT processors that support 386
- debugging stylesheets
  - example of 71
  - parser errors 489
  - processor errors 489
  - using breakpoints 480
  - what output this template generates 486
  - which template generated output 485
- debugging XQuery
  - bookmarks 728
  - displaying expressions associated with output 727
  - displaying process suspension points 727
  - displaying processing information 726
  - evaluating XPath expressions 726
  - local variables and 727
  - overview 724
  - using breakpoints 725
  - watching variables 726
- debugging XSLT
  - bookmarks and 484
  - breakpoints 480
  - determining templates associated with output 485
  - displaying instructions associated with output 484

- displaying process suspension points 483
  - displaying processing information 481
  - evaluating XPath expressions 482
  - example of 71
  - local variables and 482
  - overview 479
  - watching variables 482
  - default templates
    - description 383
    - example 30
    - how they work 344
  - demos
    - video demonstrations of Stylus Studio features 48
  - descendant axis 643
  - descendant-or-self axis 646
  - .dff files
    - default Stylus Studio module and 110
  - Diagram** tab
    - XML Schema Editor 508
  - diffing XML
    - changes that are identified 197
    - colors and symbols used by the XML Diff Viewer 200
    - command line utility for 223
    - diffing folders 201
    - diffing multiple documents 214
    - diffing two documents 213
    - merged view of diffed documents 208
    - options 219
    - overview 196
    - text view of diffed documents 207
    - tools for documents and folders 195
    - tree view of diffed documents 206
    - tuning the diffing algorithm 198
    - video demonstration 195
    - viewing documents side-by-side 206
    - when the diff is calculated 199
    - while diffing folders 205
    - XML Diff Viewer
      - adding documents 212
      - example 196
      - tool bar 209
  - displaying line numbers 8
  - document pane
    - display features for 244
  - document wizards
    - CSV to XML 156
    - custom document wizards 801
    - DTD to XML Schema 499
    - DTD to XML Schema (Trang) 499
    - EDIFACT to XML Schema 506
    - Fixed Width to XML 156
    - for converting text to XML 156
    - HTML to XSLT 39
    - XML to XML Schema 504
  - document() function
    - using 683
    - XSLT mapper and 446
  - documentation
    - for XML Schema 564
    - Stylus Studio documentation xli
  - Documentation** tab
    - XML Schema Editor 510
  - documents
    - creating backup copies 233
    - custom document wizards 801
    - saving 233
  - DTD
    - spell checking documents 171
    - using to create XML Schema 499
  - .dtd files
    - default Stylus Studio module and 110
  - dynamic text
    - inserting in three-pane view 41
  - dynamic Web pages
    - from static HTML 38
    - from XML
      - three-pane view 339
- ## E
- EDI
    - creating XML Schema from EDIFACT message types 506
  - EDI files
    - converting to XML 284
    - Convert to XML module
  - EDIFACT
    - creating XML Schema from EDIFACT message types 506

- 
- EDIFACT to XML Schema document wizard
    - built-in EDI adapter and 287
    - description 506
    - options for 506
    - running 506
    - uses for XML Schema 287
  - editing XML
    - bookmarks 169
    - changing fonts 168
    - colors used in text display 169
    - commenting text 169
    - diffing 195
    - displaying line numbers 8
    - displaying white space in schema representations 19
    - features for 166
    - indenting text 167
    - inserting indents 11
    - line wrap 167
    - querying a document 12
    - searching text 169
    - Sense:X auto-completion 9
    - setting bookmarks 169
    - spell checking documents 171
    - tools for 166
    - undo 11
    - wrapping lines 167
  - examples
    - bookstore.xml 618
    - creating dynamic Web pages from XML
      - three-pane view 339
    - creating templates 348
    - making static Web pages dynamic 38
    - stylesheet 317
    - testing queries 620
    - tree representation of XML data 618
    - video demonstrations of features 48
    - XML document structure 617
  - expand all 177
  - expanded node names
    - obtaining 676
  - extension functions
    - data types 378
    - declaring 378
    - finding 380
    - invoking 380
  - namespaces 379
  - XPath data types 379
- F**
- field names
    - specifying in Convert to XML 260
  - File Explorer
    - adding files to Projects using
    - features of 113
    - filters for 114
    - opening files with 113
    - overview 112
    - setting filters to display files in 114
    - tool bar 113
  - files
    - see also* CSV, EDI, and flat files
    - adding file types to Stylus Studio 117
    - adding to projects
    - associating file types with Stylus Studio tools 111
    - converting non-XML files to XML 235
    - custom file systems 813
    - file types and Stylus Studio module associations 110
    - making Stylus Studio the default application 117
    - opening 110
    - spell checking files in Stylus Studio 171
  - filters
    - for XQuery 627
  - Find
    - searching XML documents 232
  - fixed-width files
    - converting to XML
      - Convert to XML module 235
      - document wizard 156
  - flat files
    - converting to XML
      - Convert to XML module 235
    - creating XML documents from 238
  - floor() function 660
  - flow ports
    - in XSLT mapper symbols 460

- FLWOR blocks
  - creating 718
  - Flow port 718
  - For port 717
  - illustration 717
  - in XQuery Mapper 717
  - Order by port 717
  - Return port 717
  - Where port 717
- folders
  - diffing folder contents 201
  - Other Documents folder in Stylus Studio projects 120
- following axis 644
- following-sibling axis 644
- fonts
  - changing 168
  - colors used in text display 169
  - in Convert to XML document pane 246
- formatting objects (FO)
  - automatic tag completion 366
  - generating 392
- function blocks
  - preserving layout in XQuery mapper 711
  - preserving layout in XSLT mapper 454
- function keys
  - applying stylesheets 357
- function-available() function 680

## G

- generate-id() function 683
- Go To dialog box
  - Convert to XML Editor 246
- Grid tab
  - overview 179
  - renaming nodes in 184

## H

- handling errors in stylesheets 489
- Home Edition
  - description 3

- HTML
  - automatic tag completion 366
  - composing XSLT for 32, 369
  - creating XML documents from 163
  - creating XSLT 354

## I

- IBM DB2
  - support for 766
- id() function 671
- IDs
  - finding elements with 671
  - public IDs for XML Schemas 556
  - system IDs for XML Schemas 556
  - temporary 683
- IF blocks
  - in XQuery mapper 722
- images
  - example of dynamic images in XSLT 45
  - including in XML Schema documentation 567
- indenting tags 11
- indenting text 167
- Informix
  - support for 766
- input ports
  - in XQuery Mapper symbols 720
  - in XSLT mapper symbols 459
- INSERT statement
  - SQL/XML example 786
- instantiating templates
  - process flow 346

## J

- Java
  - compiler options 140
  - compiling code generated from XQuery 743
  - compiling code generated from XSLT 403
  - defining functions in XSLT mapper 469
  - extension functions for stylesheets 380
  - external JVM options 141
  - generating code from XQuery 739
  - generating code from XSLT 399



- generating JAXB classes from XML Schema 571
- JVM options 139
- Java API
  - Javadoc for 813
  - using to convert files 287
- Java Code Generation wizard for XQuery
  - about 739
  - compiling code from 743
  - running 742
  - settings for 741
- Java Code Generation wizard for XSLT
  - about 399
  - compiling code from 403
  - running 402
  - settings for 401
- .java files
  - default Stylus Studio module and 110
- Javadoc
  - Stylus Studio File System Java API 813
- JavaScript in stylesheet results 333
- JAXB
  - generating classes from XML Schema 571
- JRE
  - requirements
    - for DB-to-XML data sources 766
    - for Java debugging 490
- JVM
  - external JVM options 141
- JVM options 139

## K

- `key()` function 681
- keyboard shortcuts 145

## L

- `lang()` function 657
- `languages.xml` file 366
- `last()` function 678
- line numbers
  - displaying 8
  - jumping to a line in an XML document 231
- line wrap 167

- local variables
  - watching during XQuery processing 727
- `local-name()` function 676
- logical operators
  - in XSLT mapper 467

## M

- Mapper
  - for XQuery 700
  - for XSLT 439
- mapper layout
  - XQuery features for 711
  - XSLT features for 454
- Mark Logic
  - using Mark Logic to process XQuery 733
- `match` attribute
  - comparison with `select` attribute 323
  - description 321
- matched templates
  - creating in XSLT Mapper 471
- metrics
  - XQuery performance 728
  - XSLT performance 486
- Microsoft SQL Server
  - support for 766
- multidocument queries
  - alternatives 684

## N

- `name()` function 675, 676
- named templates
  - creating in XSLT Mapper 471
- namespace axis 646
- namespaces
  - extension functions 379
  - obtaining information in queries 675
  - stylesheets 320
- `namespace-uri()` function 676
- node sets
  - comparing 663
- `node()` function 672

- nodes
  - in-place editing in a schema diagram 95
  - obtaining expanded names 676
  - obtaining the local name 676
  - obtaining the namespace URI 676
  - renaming in the XML Grid tab 184
- `normalize-space()` function 652
- NOT operator 655
- `not()` function 657
- `number()` function 659
- numbers
  - converting operands to 659
  - displaying line numbers in Stylus Studio editors 8

## O

- opening files 110
  - using the File Explorer 113
- operands
  - converting to Booleans 656
  - converting to numbers 659
  - converting to strings 653
- Options dialog box
  - Custom Document Wizards page 803
  - Custom File Systems page 815
  - Custom Validation Engines page 797
- OR operator 655
- Oracle
  - support for 766

## P

- parameters in queries 679
- parent axis 643
- performance
  - factors that affect Stylus Studio 151
  - reporting XQuery metrics 728
  - reporting XSLT metrics 486
- performance metrics
  - for XQuery 728
  - for XSLT 475
- `position()` function 667
- POST data
  - using `POST.htm` to test queries 620

- post-processing XSLT 391
- preceding axis 645
- preceding-sibling axis 644
- printing
  - XML Schema 513
  - XML Schema documentation 571
- .prj files
  - default Stylus Studio module and 110
- `processing-instruction()` function 672
- profiling
  - XQuery
    - Profiler report 728
    - stylesheet for Profiler report 729
  - XSLT
    - stylesheet for Profiler report 487
- projects
  - adding files to
    - ClearCase and 129, 131
  - creating 121
  - definition 118
  - opening 121
  - placing under source control 127
  - saving 121
  - setting classpaths for 125
  - SourceSafe and 129
  - subprojects and 121
  - Visual SourceSafe and 129
  - Zeus CVS and 134
- public IDs for XML Schemas 556

## Q

- qualified names
  - wildcards 677
- queries
  - axis syntax 641
  - document element 617
  - document structure 617
  - function available? 680
  - getting started 621
  - IDs 671
  - multiple documents 684
  - non-XML data 616

- query language
  - attributes 626
  - Boolean expressions 655
  - comparisons 661
  - context flags 638
  - context nodes 635
  - context summary 640
  - count 678
  - filtering results 629
  - filters 627
  - getting a subscript 667
  - getting all marked-up text 621
  - `id()` function 671
  - namespaces 675
  - node names 675
  - obtaining all like-named elements 622
  - operators 662
  - path operators 638
  - quick reference for functions and methods 686
  - search context 635
  - searching by node type 672
  - selecting nodes to evaluate 634
  - subscripts 666
  - wildcards 630
  - wildcards in attributes 627
- restrictions 616
- root node 617
- sorting attributes 616
- subqueries 629
- temporary IDs 683
- tutorial 621
- variables 679
- where you can specify 614
- query facility 613

## R

- `.rdbxml` files
  - default Stylus Studio module and 110
- refactoring XML Schema nodes 97

- regular expressions
  - reference 232
  - using Search to find 232
  - using to filter output of converted documents 270
  - `xsdpattern` and 521
- relational data
  - composing SQL/XML 780
  - connecting to a database 773
  - DataDirect SequeLink and 767
  - defining DB-to-XML data sources 768
  - editing SQL/XML 768
  - extracting using XMLForest 790
  - how it is translated to XML 782
  - working with relational data in Stylus Studio 766
- removing templates 386
- restrictions
  - queries 616
- result documents
  - getting started with 35
- root element 617
  - definition 635
- root node
  - creating matching template 348
  - default template 384
  - definition 636
  - matching template 344
- root/element default template
  - description 384
  - example 344
- `round()` function 661

## S

- saving documents
  - automatic save 233
  - creating backup copies 233
  - options for 233
- scenarios
  - cloning 779
  - creating 777
  - definition 33
  - deleting 780
  - for DB-to-XML data sources 767, 776
  - for Web service calls 758

- for XQuery 731
  - performance metrics reporting 728
- for XSLT 33
  - choosing an XSLT processor 475
  - how to clone 478
  - how to create 476
  - how to run 477
  - introduction 472
  - performance metrics reporting 475
  - setting parameter values 474
  - specifying source documents 473
- modifying 779
- renaming 780
- XQuery
  - validating results 735
- XSLT
  - validating results 389
- search context
  - definition 635
  - queries 635
- searching
  - text 169
  - using Find to search XML documents 232
  - using XPath to search for strings 648
- select attribute
  - comparison with `match` attribute 323
  - description 322
  - when there is none 344
- SELECT statement
  - SQL/XML example 785
- self axis 646
- Sense:X automatic tag completion
  - description 9
  - example 10
- Sense:X tag completion
  - XSLT and 366
- SequeLink
  - DataDirect SequeLink and Stylus Studio 767
- Sleepycat Software
  - integration with Stylus Studio 817
  - opening files stored on Berkeley DB XML 824
- SOAP requests
  - modifying 752
  - parameters for 752
- source control
  - Stylus Studio projects and 127
  - supported applications 128
- SourceSafe
  - using with Stylus Studio 129
- Spell Checker
  - personal dictionary 174
  - running 173
  - settings for 172
  - using 171
- SQL/XML
  - composing 780
  - editing in Stylus Studio 768
  - INSERT statement example 786
  - SELECT statement example 785
- Sruzzo
  - command line utility for running Stylus Studio 148
- `string()` function 653
- `string-length()` function 651
- strings
  - after 650
  - before 649
  - concatenating 651
  - converting operands to 653
  - number of characters 651
  - replacing characters 652
  - searching for 648
  - substrings 650
- stylesheets
  - applied by XSLT processors 323
  - applying
    - Stylus Studio 357
  - chaining 148
  - contents 320
  - contents description 364
  - creating 353
  - creating new nodes 330
  - example 317
  - for XML Schema documentation 565
  - for XQuery Profiler reports 729
  - for XSLT Profiler reports 487
  - formatting results 329

- getting started with 26
  - introduction 317
  - namespaces 320
  - obtaining system properties 679
  - omitting source data 327
  - root element 320
  - selecting nodes for processing 325
  - Stylus Studio
    - debugging 479
    - updating
      - three-pane view 365
    - XSLT instructions 405
  - Stylus Studio
    - associating file types with 111
    - benefits 335
    - command line for running 148
    - command line utilities for 147
    - DataDirect SequeLink and 767
    - debugging stylesheets 479
    - Diff tool 195
    - Home Edition description 3
    - managing performance 151
    - projects 118
    - running from the command line 148
    - system requirements for DB-to-XML data
      - sources 766
    - using SQL/XML 780
    - using with ClearCase 129, 131
    - using with SourceSafe 129
    - using with Visual SourceSafe 129
    - using with Zeus CVS 134
    - video demonstrations 48
    - Web services and 745
    - XML validation command line utility 150
    - XQuery command line utility for 149
    - XSLT command line utility 148
  - Stylus Studio Home Edition
    - description 3
  - StylusDiff 223
  - StylusValidator
    - command line XML validation utility 150
  - StylusXq1
    - command line XQuery utility 149
  - StylusXslt
    - command line XSLT utility 148
  - substring() function 650
  - substring-after() function 650
  - substring-before() function 649
  - sum() function 660
  - support, technical xli
  - Sybase
    - support for 766
  - symbols
    - element and attribute
      - in XQuery mapper 707
      - in XSLT mapper 451
    - lines linking nodes in XQuery mapper 713
    - XLST function blocks
      - parts of 464
    - XLST mapper
      - parts of 458
    - XQuery function blocks
      - parts of 720
    - XQuery mapper
      - document 704
    - XSLT mapper
      - document 448
      - XSLT instructions 457
  - syntax, notations used in this manual xl
  - system IDs for XML Schemas 556
  - system properties in stylesheets 679
  - system-property() function 679
- ## T
- technical support xli
  - templates
    - applying 386
    - backmapping in 485
    - built-in 328
    - contents description 321
    - creating 385
    - creating named and matched templates in
      - XSLT Mapper 471
    - description of default templates 383
    - displaying match patterns 28
    - instantiation example 344
    - introduction 320
    - match attribute 323
    - matched templates in XSLT Mapper 471

- matching root node 348
- more than one match 328
- named templates in XSLT Mapper 471
- no match 328
- removing 386
- rules 322
- select attribute 323
- selecting for instantiation 322
- updating 386
- viewing 381
- working with in XSLT mapper 470
- text
  - colors used to display 169
- text files
  - converting to XML
    - Convert to XML module 235
- text pane
  - XML Schema Editor 509
- text values
  - setting for elements and attributes in XQuery mapper 710
  - setting for elements and attributes in XSLT mapper
    - introduction 467
    - on the target node 469
    - using the mapper canvas 468
- `text()` function 672
- text/attribute default template
  - description 384
  - example 346
- three-pane view in Stylus Studio
  - result documents 35
- tool bars
  - changing appearance of Stylus Studio main tool bar 138
  - customizing Stylus Studio main tool bar 136
  - File Explorer tool bar 113
  - showing and hiding Stylus Studio main tool bar 137
  - XML Diff Viewer tool bar 209
- `translate()` function 652
- Tree** tab
  - XML Schema Editor 509
- tutorial for queries 621
- typographical conventions xl

## U

- UDDI registries
  - searching 749
  - Web services and 748
- UN/CEFACT
  - creating XML Schema from EDIFACT message types 506
- `unparsed-entity-uri()` function 678
- updating stylesheets 365
- user-defined field names
  - display in Convert to XML Editor 243
- user-defined functions
  - in XQuery mapper 721

## V

- validating XML
  - custom validation engines for 795
  - from the command line 150
  - standard validation engines for 795
- variables in queries 679
- 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
  - Convert to XML 235
  - DB-to-XML data sources 765
  - diffing XML sources 195
  - of Stylus Studio features 48
  - Web Service Call Composer 745
  - XML Editor Grid tab 180
  - XQuery Mapper 695
  - XSLT WYSIWYG 47
- viewing templates 381
- Visual SourceSafe
  - using with Stylus Studio 129

## W

- Web pages
  - changing from static to dynamic HTML 38
  - creating from XML
    - three-pane view 339
- Web Service Call Composer
  - video demonstration 745
- Web service calls
  - specifying transport protocols for 761
- Web services
  - creating a Web service call 746
  - saving Web service calls 755
  - scenarios for Web service calls 758
  - SOAP requests for 752
  - testing 754
  - using in Stylus Studio 745
  - using Web service calls as XML 756
  - WSDLs and 748
- white space
  - handling
    - XPath processor 652
    - XSL facility 330
  - toggling display in schema representations 19
- wildcards
  - in queries 630
  - node names 675
- wizards
  - custom document wizards 801
- wrapping lines 167
- .wsc and .wsc files
  - default Stylus Studio module and 110
- WSDLs
  - displaying a WSDL document 753
  - finding WSDL URLs 748
  - searching UDDI registries for 749
- WYSIWYG
  - XSLT editor for composing HTML 32, 369

## X

- `xln:text()` function 654
- XML 618
  - command line utility for validating 150
  - converting EDI to XML 284

- converting files programmatically 287
- converting XML to canonical XML 229
- creating from HTML 163
- creating XML documents from flat files 238
- creating XML Schema from an XML
  - document 504
- custom validation engines 795
- diffing in Stylus Studio 195
- displaying the XML Schema associated with a
  - document 506
- representing relational data as 782
- spell checking documents 171
- standard validation engines for 795
- using Web service calls as XML 756
- viewing sample XML based on XML
  - Schema 512
- XML Diff Viewer
  - adding documents 212
  - example 196
  - merged view 208
  - options 219
  - split view 206
  - text view 207
  - tool bar 209
  - tree view 206
- XML documents
  - comparing 195
  - converting to canonical XML 229
  - creating from HTML 163
  - diffing 195
  - jumping to a line in 231
  - jumping to a matching tag 232
  - querying using XPath 229
  - root element 617
  - root node 617
  - searching 232
  - setting bookmarks in 231
  - structure 617
  - tools for diffing 195
  - XSLT 614
- XML Editor
  - displaying line numbers in 8
  - Grid tab
    - overview 179
    - renaming nodes in 184
  - jumping to a line 231

- jumping to a matching tag 232
- searching 232
- setting bookmarks 231
- XML Editor Grid tab
  - video demonstration 180
- .xml files
  - default Stylus Studio module and 110
- XML instances
  - XQuery mapper source documents and 702
  - XSLT mapper source documents and 446
- XML Schema
  - creating 498
  - creating from an XML document 504
  - creating from DTD 499
  - creating from EDIFACT message types 506
  - definition 498
  - detecting errors in 93
  - Diagram** tab
    - description 508
    - illustration 508
  - displaying documentation element text in the Diagram tab 91
  - displaying documentation using XS3P stylesheet 565
  - displaying documentation using XSDdoc stylesheet 569
  - displaying the XML Schema associated with a document 506
  - documentation for 564
  - Documentation** tab
    - description 510
  - generating JAXB classes from 571
  - nodes
    - refactoring 97
  - printing 513
  - refactoring nodes 97
  - reference information 498
  - spell checking documents 171
  - Stylus Studio tools for 508
  - Tree** tab
    - description 509
  - validating 511
  - viewing sample XML for 512
- XML Schema diagram
  - in-place editing 95
- XML Schema documentation
  - printing 571
- XML Schema Editor
  - displaying errors in text pane 93
  - text pane 509
- XML to XML Schema document wizard 504
- XMLForest
  - support for 790
- XPath
  - background 614
  - benefits 615
  - choosing a version 12
  - evaluating expressions during XQuery processing 726
  - function blocks in XSLT mapper
    - creating 466
    - deleting 466
    - introduction 464
    - types 465
  - mathematical function blocks in XSLT mapper 466
  - support for 12
- XQuery
  - command line utility for 149
  - compiling Java code generated from 743
  - `concat` function blocks in XQuery mapper 722
  - creating using the XQuery Mapper 700
  - debugging XQuery documents 724
  - editor for
    - description 696
    - Mapper tab 699
    - XQuery Source tab 697
  - enabling the Profiler 730
  - evaluating XPath expressions during XQuery processing 726
  - generating Java code from 739
  - in Stylus Studio 696
  - performance metrics reporting 728
  - profiling 728
  - scenarios for 731
  - selecting a processor 733
  - spell checking documents 171
  - using existing XQueries in the XQuery editor 701



- validating result documents 735
- viewing source code in the mapper 699
- W3C definition 696
- XQuery debugging
  - bookmarks 728
- XQuery documents
  - setting breakpoints in 725
- .xquery files
  - default Stylus Studio module and 110
- XQuery Mapper
  - creating XQuery 700
  - input ports in Mapper symbols 720
  - using 701
  - video demonstration 695
- XQuery mapper
  - adding source documents 704
  - building a target structure 708
  - choosing source documents 702
  - condition blocks 723
  - creating FLWOR blocks 718
  - creating function blocks 719
  - creating target structure elements and attributes 709
  - creating target structure root elements 708
  - document symbols 704
  - element and attribute symbols in 707
  - elements and attributes
    - creating in target structures 709
  - FLWOR block parts 717
  - function blocks
    - about 719
    - creating 719
    - parts of 720
    - types 719
  - how documents are displayed 706
  - how to map nodes 712
  - IF blocks 722
  - lines linking nodes 713
  - mapping document nodes 711
  - modifying the target structure 709
  - preserving mapper layout 711
  - removing a node from a target structure 710
  - removing node mappings 716
  - removing source documents 706
  - root elements
    - creating in target structures 708
    - setting text values 710
  - source documents and XML instances 702
  - source documents for 701
  - target structures 707
  - user-defined functions 721
  - using FLWOR blocks 716
  - using the mouse 711
  - viewing source code in 699
- XQuery output
  - displaying source expressions 727
- XQuery processing
  - call stack 727
  - displaying source expressions 727
  - displaying suspension points 727
  - using bookmarks 728
  - watching local variables 727
  - watching variables 726
- XQuery processors
  - Mark Logic 733
  - selecting 733
- XQuery Profiler
  - description 728
  - displaying the report 731
  - enabling 730
  - performance metrics captured by 729
  - report created by 729
- XQuery scenarios
  - performance metrics reporting 728
- XS3P stylesheet
  - display settings 567
  - displaying XML Schema documentation
    - with 565
  - features 566
  - modifying 568
- XSD
  - displaying documentation element text 91
- .xsd files
  - default Stylus Studio module and 110
- xsd:pattern
  - regular expressions and 521
- XSSdoc
  - displaying XML Schema documentation
    - with 569

- XSDdoc stylesheet
  - display settings 569
- XSL
  - additional information sources 334
  - and XSLT 614
  - definition 316
  - example 317
  - formatting objects 392
  - getting started with 315
  - inserting JavaScript in result 333
  - patterns 331
- xsl
  - choose
    - editing in XSLT mapper 463
- XSL facility
  - creating new nodes 330
  - selecting source nodes 325
  - specifying XSL patterns 331
  - white space handling 330
- .xsl files
  - default Stylus Studio module and 110
- XSL processor
  - applying stylesheets 323
  - built-in templates 328
  - specifying result format 329
  - URI 320
- xsl:apply-imports instruction 406
- xsl:apply-templates instruction
  - comparison with xsl:for-each instruction 333
  - controlling operation order 326
  - example 326
  - more than one match 328
  - no match 328
  - no select attribute 344
  - reference 406
  - selecting nodes 325
  - specifying patterns 331
- xsl:attribute instruction 407
- xsl:attribute-set instruction 409
- xsl:call-template instruction 411
- xsl:choose
  - compared to xsl:if in XSLT mapper 462
- xsl:choose instruction 411
- xsl:comment instruction 412
- xsl:copy instruction 413
- xsl:copy-of instruction 414
- xsl:debug instruction 480
- xsl:decimal-format instruction 414
- xsl:element instruction 416
- xsl:fallback instruction 417
- xsl:for-each instruction
  - comparison with xsl:apply-templates instruction 333
  - reference 417
  - selecting nodes 325
  - specifying patterns 331
- xsl:if
  - compared to xsl:choose in XSLT mapper 462
- xsl:if instruction 419
- xsl:import instruction
  - reference 420
- xsl:include instruction 420
- xsl:key instruction 421
- xsl:message instruction 422
- xsl:namespace-alias instruction 423
- xsl:number instruction 423
- xsl:otherwise instruction 425
- xsl:output instruction
  - controlling white space 331
  - reference 425
- xsl:param instruction 427
- xsl:preserve-space instruction 428
- xsl:processing-instruction instruction 428
- xsl:sort instruction 429
- xsl:strip-space instruction 431
- xsl:stylesheet instruction
  - reference 432
- xsl:template instruction
  - creating new nodes 330
  - reference 432
  - specifying patterns 331
- xsl:text instruction
  - creating white space 330
  - reference 434
- xsl:transform instruction 435
- xsl:value-of instruction
  - reference 435
  - specifying patterns 331
- xsl:variable instruction 436
- xsl:vendor property 679

- xsl:vendor-url property 679
- xsl:version property 679
- xsl:when instruction 437
- xsl:with-param instruction 437
- XSLT
  - automatic tag completion 366
  - background 316
  - chaining stylesheets 148
  - choosing an XSLT processor 475
  - command line utility for 148
  - compiling Java code generated from 403
  - composing for HTML 32, 369
  - creating from HTML 354
  - creating using the XSLT Mapper 439
  - default processor settings 388
  - generating formatting objects 393
  - generating Java code from 399
  - instruction blocks in XSLT mapper 460
  - introduction 317
  - performance metrics reporting 475
  - post-processing result documents 391
  - processors
    - default options for 388
  - spell checking documents 171
  - symbols used to represent instructions in XSLT mapper 457
  - tags reference 405
  - validating result documents 389
  - viewing source code in the mapper 440
- XSLT Editor
  - displaying line numbers in 8
- XSLT Mapper
  - creating XSLT 439
- XSLT mapper
  - adding instruction blocks 460
  - adding source documents 448
  - building a target structure 452
  - choosing source documents 445
  - creating target structure elements and attributes 453
  - creating target structure root elements 452
  - defining Java functions in 469
  - document symbols 448
  - element and attribute symbols in 451
  - elements and attributes
    - creating in target structures 453
  - example 441
  - flow ports in mapper symbols 460
  - how documents are displayed 450
  - how to map nodes 456
  - input ports in mapper symbols 459
  - logical operators 467
  - mapping document nodes 454
  - modifying the target structure 453
  - options for 443
  - overview 439
  - overview of creating XSLT 444
  - preserving mapper layout 454
  - processing source nodes 464
  - removing a node from a target structure 454
  - removing node mappings 456
  - removing source documents 450
  - root elements
    - creating in target structures 452
  - setting text values
    - introduction 467
    - on the target node 469
    - using the mapper canvas 468
  - source documents and document ()
    - function 446
  - source documents and XML instances 446
  - source documents for 445
  - support for XSLT instructions and expressions 442
  - symbols for XSLT functions
    - parts of 464
  - symbols for XSLT instructions
    - list 457
    - parts of 458
  - target structures 451
  - using the mouse 455
  - viewing source code in 440
  - working with templates 470
- XPath function blocks
  - creating 466
  - deleting 466
  - introduction 464
  - types 465

- XPath mathematical function blocks 466
- XSLT processors
  - backmapping and 386
  - choosing 387
  - debugging and 386
- XSLT Profiler
  - displaying the report 489
  - enabling 488
  - performance metrics captured by 488
- XSLT scenarios
  - choosing an XSLT processor 475
  - cloning 478
  - creating
    - how to 476
    - introduction 472
  - performance metrics reporting 475
  - running 477
  - setting parameter values 474
  - specifying source documents 473
- XSLT stylesheets
  - debugging 479
- XSLT WYSIWYG
  - video demonstration 47

## Z

- Zeus CVS
  - using with Stylus Studio 134